

Tutorial on
Validated Scientific Computing
Using Interval Analysis

George F. Corliss
Marquette University
Milwaukee, Wisconsin
George.Corliss@Marquette.edu
www.eng.mu.edu/corlissg/PARA04

PARA'04
Workshop on State-of-the-Art
in Scientific Computing
Technical University of Denmark
June 20-23, 2004

Validated Scientific Computing Using Interval Analysis

Target audience

- Interval beginners, e.g., graduate students, applications researchers
- Assume first course in scientific computation, e.g.,
 - Floating point arithmetic
 - Error analysis
 - Automatic differentiation?
 - Gaussian elimination
 - Newton’s method
 - Numerical optimization
 - Runge-Kutta

Goals of this tutorial

- How, not why
- Define f to use in existing software
- Read modern papers on validating algorithms
- Adapt and extend them for your own use
- Write your own algorithms and papers

We **will** fall short

Talk about solutions or challenges?

Using an interval tool should be easy

Developing it is not

Outline

Similar to a standard introduction to scientific computing:

- Arithmetic and errors - Interval Arithmetic
- Linear systems
- Nonlinear root finding
- Bounding ranges of functions
- Global optimization
- Quadrature
- Ordinary differential equations
- (Partial differential equations)
- Lessons learned
- References

We **will not** finish

Abstract

This tutorial introduces interval beginners (e.g., graduate students and applications researchers) to concepts and patterns of interval analysis. We assume a first course in scientific computation typically including floating point arithmetic, error analysis, automatic differentiation, Gaussian elimination, Newton’s method, numerical optimization, and Runge-Kutta methods for ODE’s. At the conclusion of this tutorial, you should be able to

- Understand most of the how and why of interval algorithms
- Define your own f to use in existing software
- Read modern papers on validating algorithms
- Adapt and extend them for your own use
- Write your own algorithms and papers

Coverage in this tutorial is similar to a standard introduction to scientific computing:

- Arithmetic and errors - Interval Arithmetic
- Linear systems
- Nonlinear root finding
- Bounding ranges of functions
- Global optimization
- Quadrature
- Ordinary differential equations
- Partial differential equations
- Lessons learned
- References

We will not finish all these topics. The tutorial will include a lecture with examples in MATLAB and Sun’s Fortran 95 and a set of supervised, hands-on exercises.

Can you suggest some readings about interval arithmetic to help me get started?

Some of my favorites:

- R. E. Moore**, *Interval Analysis*, Prentice Hall, Englewood Cliffs, N.J., 1966. The bible (little b)
- R. E. Moore**, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, Penn., 1979
- G. Alefeld and J. Herzberger**, *Einführung in die Intervallrechnung*, Springer-Verlag, Heidelberg, 1974
- Introduction to Interval Computations*, Academic Press, New York, 1983
- A. Neumaier**, *Interval Methods for Systems of Equations*, Cambridge University Press, 1990
- E. R. Hansen**, *Global Optimization Using Interval Analysis*, Marcel Dekker, New York, 1992
- R. Hammer, M. Hocks, U. Kulisch, and D. Ratz**, *C++ Toolbox for Verified Computing I: Basic Numerical Problems: Theory, Algorithms, and Programs*, Springer-Verlag, Berlin, 1995
- Baker Kearfott**, *Rigorous Global Search: Continuous Problems*, Kluwer, 1996
- G. F. Corliss**, SCAN-98 collected bibliography, in *Developments in Reliable Computing*, T. Csendes, ed., Kluwer, Dordrecht, Netherlands, 1999
- Gamma, Helm, Johnson, and Vlissides** (GOF - “Gang of Four”), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Mass., 1995

On the web:

- N. Beebe and J. Rokne, *Bibliography on Interval Arithmetic*, linwww.ira.uka.de/bibliography/Math/intarith.html
- O. Caprani, K. Madsen, and H. Nielsen, *Introduction to Interval Analysis*, www.imm.dtu.dk/courses/02611/IA.pdf
- V. Kreinovich, D. Berleant, M. Koshelev, *Interval Computations*, www.cs.utep.edu/interval-comp
- A. Neumaier, *Global Optimization*, www.mat.univie.ac.at/~neum/glopt.html
- Sun Microsystems, An exciting and enthusiastic push for intervals on a recent Sun posting featuring our own Bill Walster: www.sun.com/presents/minds/2004-0527/

Applications?

“Consider ...”
— Pure math

“This could be used for ...”
— Applicable math

$n/2$ slides of science
1 slide of math or methods
 $n/2$ slides of science
— APPLIED math

This is APPLICABLE talk

Design Patterns in Validated Computing

Ref: GOF: Gamma, Helm, Johnson, and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*
Encapsulate the concept that varies. Probably the most influential (on me) technology book I’ve read in a decade.

Corliss’s design patterns in validated computing:

- Interval arithmetic captures modeling, round-off, and truncation errors
- Simplify first. Do arithmetic at last moment. Use SUE
- Interval I/O is subtle/interesting/hard
- Good floating-point algorithms rarely make good interval algorithms
- Contractive mapping theorems are the basis for effective computational algorithms for validating existence and uniqueness of solutions
- Know what is interval and what is point
- Use Mean Value, centered form, or slope representations
- If you can readily compute two enclosures, intersect

- Approximate, then validate

Interval Arithmetic

Pattern: Interval arithmetic captures modeling, round-off, and truncation errors

Think of Interval as an abstract data type in object-oriented sense if C++, Java, Fortran 95:

```
class Interval {
    private representation;
    // Utilities
    Interval(); Inf(); Sup(); ...
    // Arithmetic operations
    +, -, *, /
    // Elementary functions
    sin(), cos(), sqrt(), log(), exp(), ...
    // Set and Logical operations
    =, !=, ...
    Certainly less than, Possibly less than, ...
    Contains, Properly contains, ...
    ...
}
```

Operations are defined setwise
 $\mathbf{x} \circ \mathbf{y} := \{\tilde{x} \circ \tilde{y} : \tilde{x} \in \mathbf{x}, \tilde{y} \in \mathbf{y}\}$

Interval Arithmetic - Exact Arithmetic

Assume computer arithmetic is exact (laugh track ...)

Operations are defined setwise
 $\mathbf{x} \circ \mathbf{y} := \{\tilde{x} \circ \tilde{y} : \tilde{x} \in \mathbf{x}, \tilde{y} \in \mathbf{y}\}$
Notation: $\mathbf{x} = [\underline{x}, \bar{x}] = [\text{Inf}(\mathbf{x}), \text{Sup}(\mathbf{x})]$

Add: Ex., $[3, 4] + [-7, -4] = [?, ?]$
 $\mathbf{x} + \mathbf{y} = [\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$

Multiply: Ex., $[3, 4] * [-7, -4] = [?, ?]$
9 cases depending on signs, or
 $\mathbf{x} * \mathbf{y} = [\underline{x}, \bar{x}] * [\underline{y}, \bar{y}]$
 $= [\min(\underline{x} * \underline{y}, \underline{x} * \bar{y}, \bar{x} * \underline{y}, \bar{x} * \bar{y}), \max(\underline{x} * \underline{y}, \underline{x} * \bar{y}, \bar{x} * \underline{y}, \bar{x} * \bar{y})]$

Lab Exercise 1. Interval Operations

Exercise 1.1. Write a pseudocode function for interval subtraction $\mathbf{x} - \mathbf{y} =$

[In any lab exercise, some people finish before others. In each exercise, if you finish part 1 ahead of your colleagues, I encourage you either help your colleagues or to go on to following parts:]

Exercise 1.2. Write a pseudocode function for interval division $x/y =$

Exercise 1.3. Write a pseudocode function for interval EXP $\exp(x) =$

Exercise 1.4. (harder) Write a pseudocode function for interval SIN $\sin(x) =$

Take-home: Interval operations are defined setwise

Take-home: Most, but not all, interval operations depend only on endpoints.

Take-home: Beware of those that do not.

Interval Arithmetic - History

Ramon Moore – “Father of interval analysis?”

Archimedes used two-sided bounds to compute Pi: Archimedes, “On the measurement of the circle,” In: Thomas L. Heath (ed.), *The Works of Archimedes*, Cambridge University Press, Cambridge, 1897; Dover edition, 1953.

M. Warmus, Calculus of Approximations, Bull. Acad. Polon. Sci., Cl. III, vol. IV, No. 5 (1956), 253-259.

T. Sunaga, Theory of interval algebra and its application to numerical analysis, In: *RAAG Memoirs*, Ggujutsu Bunken Fukuy-kai. Tokyo, Vol. 2, 1958, pp. 29-46 (547-564).

R. E. Moore, Automatic error analysis in digital computation. Technical Report Space Div. Report LMSD84821, Lockheed Missiles and Space Co., 1959.

An excellent description of Moore’s contribution to interval computations from The Moore Prize for Applications of Interval Analysis

interval.louisiana.edu/Moore_prize.html:

“The idea of arithmetic over sets to encompass finiteness, roundoff error, and uncertainty dates back to the first part of the twentieth century or earlier. By the late 1950’s, with exponentially increasing use of digital electronic computers for mathematical computations, interval arithmetic was a concept whose time had come. With his 1962 dissertation *Interval Arithmetic and Automatic Error Analysis in Digital Computing*, encouraged by George Forsythe, Prof. Ramon Moore was one of the first to publicize the underlying principles of interval arithmetic in their modern form. Prof. Moore subsequently dedicated much of his life to furthering the subject. This includes guidance of seven Ph.D. students, interaction with other prominent figures in the area such as Eldon Hansen, Louis Rall, and Bill Walster, and publication of the seminal work *Interval Analysis* (Prentice Hall, 1966) and its update *Methods and Applications of Interval Analysis* (SIAM, 1979). In addition, Prof. Moore published a related book *Computational Functional Analysis* (Horwood, 1985), and

organized the conference with proceedings *Reliability in Computing* (Academic Press, 1988). This latter conference was a major catalyst for renewed interest in the subject. It is safe to say that these accomplishments of Professor Moore have made interval analysis what it is today.”

See also www.cs.utep.edu/interval-comp/early.html and interval.louisiana.edu/Moores_early_papers/bibliography.html

Example: ex1_interval.f95

Sun Fortran 95

Simple interval declaration and arithmetic

```
PROGRAM EX1_INTERVAL
  INTERVAL :: A = [3, 4], B = [-7, -4]
  PRINT *, "interval A = ", A
  PRINT *, "interval B = ", B
  PRINT *, ""

  PRINT *, "A + B: ", A + B
  PRINT *, "A * B: ", A * B
END PROGRAM EX1_INTERVAL
```

Result:

```
$ ex1
interval A = [3.0,4.0]
interval B = [-7.0,-4.0]

A + B: [-4.0,0.0E+0]
A * B: [-28.0,-12.0]
```

Take-home: Interval programming is easy

Example: ex1_interval.m

TU Hamburg-Harburg INTLAB for MATLAB
www.ti3.tu-harburg.de/english/index.html
Simple interval declaration and arithmetic

```
intvalinit('DisplayInfSup')
a = intval( '[3, 4]' )
b = intval( '[-7, -4]' )

disp( ' ' ); disp( 'a + b:'); x = a + b
disp( 'a * b:'); x = a * b
```

Result (lightly edited):

```
>> ex1_interval
====> Default display of intervals by infimum/supremum
      (e.g. [ 3.14 , 3.15 ])
intval a = [ 3.0000, 4.0000]
intval b = [-7.0000, -4.0000]
```

```

a + b:
intval x = [ -4.0000, 0.0000]
a * b:
intval x = [ -28.0000, -12.0000]

```

Example: ex1_interval.cc

Sun's C++
Simple interval declaration and arithmetic

```

#include <suninterval.h>
#if __cplusplus >= 199711
    using namespace SUNW_interval;
#endif

int main () {
    interval<double> a( "[3, 4]" );
    interval<double> b( "[-7, -4]" );
    cout << "a = " << a << endl;
    cout << "b = " << b << endl << endl;

    cout << "a + b: " << a + b << endl;
    cout << "a * b: " << a * b << endl;
    return 0;
}

```

Result:

```

$ ex1
a = [0.3000000000000000E+001,0.4000000000000000E+001]
b = [-.7000000000000000E+001,-.4000000000000000E+001]

a + b: [-.4000000000000000E+001,0.0000000000000000E+000]
a * b: [-.2800000000000000E+002,-.1200000000000000E+002]

```

Lab Exercise 2. Interactive Matlab

On the page

www.student.dtu.dk/~g04120/interval/intlab.html is a link to `startintlab.m` which is to be downloaded.

On the same page is a link to a copy of the Matlab/Intlab programs, having the name `Matlab.zip` such that they also can be downloaded, and un-zip'ed, such that each participant has a copy.

For help getting started with Matlab, see

www.student.dtu.dk/~g04120/interval/intlab.html and the link [INTLAB for Bicyclists]

Exercise 2.1. Start Matlab. In Matlab's interactive mode, type

```

>> intvinit('DisplayInfSup')
>> a = intval( '[3, 4]' )
>> b = intval( '[-7, -4]' )
>> disp( ' ' ); disp( 'a + b:'); x = a + b
>> disp( 'a * b:'); x = a * b

```

You should see something like:

```

==> Default display of intervals by infimum/supremum
      (e.g. [ 3.14 , 3.15 ])
intval a = [ 3.0000, 4.0000]
intval b = [ -7.0000, -4.0000]

a + b:
intval x = [ -4.0000, 0.0000]
a * b:
intval x = [ -28.0000, -12.0000]

```

Exercise 2.2. (Optional) Type some of the commands

```

>> intvinit('Display_')
>> intvinit('DisplayMidrad')
>> help intvinit
>> format long
>> format short e
>> help format
>> help intval

```

and repeat Exercise 2.1

Exercise 2.3. Try some more interesting expressions.

Take-home: Analog of "Hello, World." You can do interval arithmetic in Matlab.

Getting Started at DTU Databar

www.student.dtu.dk/~g04120/interval/intlab.html

Get `startintlab.m`

Log in

On desktop, left click, Programs, Netscape

URL:

www.student.dtu.dk/~g04120/interval/intlab.html

Click link `startintlab.m`

Save as, save to your root

Get Examples

From URL:

www.student.dtu.dk/~g04120/interval/intlab.html

Click link for `Matlab.zip`

Save to your root

In file cabinet, select `Matlab.zip`, Select menu, unzip

Should create a directory `Matlab`

Start Matlab

Desktop, Middle mouse button, Mathematics, Matlab

```
>> startintlab
```

```
>> cd Matlab
```

```
>> ls % Should see example files
```

```
>> ex1_interval
```

For help getting started with Matlab, see

www.student.dtu.dk/~g04120/interval/intlab.html

and the link [INTLAB for Bicyclists]

Interval Arithmetic - Exact

Arithmetic

Pattern: Simplify first.

Do arithmetic at the last moment

Also known as the “dependency problem”

$[1, 2] - [1, 2] = [?, ?]$

Depends?

```
Interval A, B, X;
A = Interval (1, 2);
X = A - A; // 0 = a - a for all a in A
B = Interval (1, 2);
X = A - B; // a - b need NOT be zero
B = A;
X = A - B;
```

In interval arithmetic

$A - A \text{ NOT } = 0$

$A * (B + C) \subseteq A * B + A * C$

A little mathematics beats a lot of computing

Whenever possible, use Single Use Expression

SUE, pronounced *soo* ^{EE} (Walster)

Theorem (Moore) Interval evaluation of a rational function

SUE is tight

Example: ex2_depend.f95

Interval dependencies: $A - A \neq 0$, sub-distributive

```
PROGRAM EX2_DEPENDENCIES
  INTERVAL :: A = [1, 2], B = [1, 2], C, X, Y
  PRINT *, "interval A = ", A
  PRINT *, "interval B = ", B
  PRINT *, ""

  PRINT *, "A - A: ", A - A, "; A .DSUB. A: ", A .DSUB. A
  PRINT *, "A - B: ", A - B, "; A .DSUB. B: ", A .DSUB. B

  PRINT *, ""
  PRINT *, "_SUB_distributive law holds:"
  A = [1, 2]; PRINT *, "interval A = ", A
  B = [5, 7]; PRINT *, "interval B = ", B
  C = [-2, -1]; PRINT *, "interval C = ", C

  X = A*(B + C)
  Y = A*B + A*C
  PRINT *, ""
  PRINT *, "X = a*(b + c): ", X
  PRINT *, "Y = a*b + a*c: ", Y
  PRINT *, "X .EQ. Y: ", X .EQ. Y
  PRINT *, "X .SB. Y: ", X .SB. Y

  PRINT *, ""
  A = [-0.1, 2]; PRINT *, "interval A = ", A
  PRINT *, "  A * A : ", A * A
  PRINT *, "  A**2 : ", A**2
END PROGRAM EX2_DEPENDENCIES
```

Result:

```
$ ex2
interval A = [1.0,2.0]
interval B = [1.0,2.0]

A - A: [-1.0,1.0] ; A .DSUB. A: [0.0E+0,0.0E+0]
A - B: [-1.0,1.0] ; A .DSUB. B: [0.0E+0,0.0E+0]

_SUB_distributive law holds:
interval A = [1.0,2.0]
interval B = [5.0,7.0]
interval C = [-2.0,-1.0]

X = a*(b + c): [3.0,12.0]
Y = a*b + a*c: [1.0,13.0]
X .EQ. Y: F
X .SB. Y: T

interval A = [-0.10000000000000001,2.0]
  A * A : [-0.20000000000000002,4.0]
  A**2 : [0.0E+0,4.0]
```

Take-home: Interval programming is **not** so easy

Example: ex2_depend.m

Repeating in Matlab, ...

Simple interval declaration and arithmetic

```
intvalinit('DisplayInfSup')
a = intval( '[1, 2]' )
b = intval( '[1, 2]' )

disp(' '); disp('x = a - a:'); x = a - a
disp(' '); disp('x = a - b:'); x = a - b

disp(' '); disp('_SUB_distributive law holds:')
a = intval( '[1, 2]' )
b = intval( '[5, 7]' )
c = intval( '[-2, -1]' )

disp(' ');
disp('x = a*(b + c):'); x = a*(b + c)
disp('a*b + a*c :'); y = a*b + a*c
disp('x == y :'); x == y
disp('in (x, y) :'); in (x, y)

disp(' ');
a = intval( '[-0.1, 2]' )
disp('a * a:'); a * a
disp('a^2:'); a^2
disp('sqrt(a):'); sqrt(a)
```

Result (lightly edited):

```
>> ex2_depend
intval a = [ 1.0000, 2.0000]
intval b = [ 1.0000, 2.0000]
```

```

x = a - a: intval x = [ -1.0000, 1.0000]
x = a - b: intval x = [ -1.0000, 1.0000]

_SUB_distributive law holds:
intval a = [ 1.0000, 2.0000]
intval b = [ 5.0000, 7.0000]
intval c = [ -2.0000, -1.0000]

x = a*(b + c): intval x = [ 3.0000, 12.0000]
a*b + a*c : intval y = [ 1.0000, 13.0000]
x == y : ans = 0
in (x, y) : ans = 1

intval a = [ -0.1001, 2.0000]
a * a : intval ans = [ -0.2001, 4.0000]
a^2 : intval ans = [ 0.0000, 4.0000]
sqr(a): intval ans = [ 0.0000, 4.0000]

```

Lab Exercise 3. Equivalent Mathematical Expressions

Exercise 3.1. In Matlab, run `ex2_depend.m`:

```
>> ex2_depend
```

Exercise 3.2. Make a copy of `ex2_depend.m`. Edit the copy to compare values of the expressions

$$\frac{x}{1+x} \text{ vs. } \frac{1}{1+\frac{1}{x}}$$

Domains?

Exercise 3.3. (Optional) Can you rearrange some of the following expressions to get narrower result intervals? (From Cheney and Kincaid, *Numerical Mathematics and Computing, Fourth Edition*, pp. 82-84)

1. $\frac{\tan x - \sin x}{x - \sqrt{1+x^2}}$
2. $\exp x - \sin x - \cos x$
3. $\ln(x) - 1$
4. $\log(x) - \log(1/x)$
5. $x^{-2}(\sin x - \exp x + 1)$
6. $x - \operatorname{arctanh} x$
7. $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Take-home: Analog of “Hello, World.” You can do interval arithmetic scripts in Matlab.

Take-home: Interval arithmetic is not real arithmetic.

Interval Arithmetic - Exact Arithmetic

Fundamental Theorem of Interval Analysis:

If a function is evaluated using interval arithmetic instead of floating-point arithmetic, the resulting interval is guaranteed to enclose the range of function values.

Rosenbrock function $f(x, y) := 100 * (y - x^2)^2 + (1 - x)^2$
 $\mathbf{X} := (\mathbf{x}, \mathbf{y}) := ([0.7, 1], [0.0, 0.3])$

For $x \in \mathbf{x}$ and $y \in \mathbf{y}$,

$$\begin{aligned}
 f(x, y) &= 100 * (y - x^2)^2 + (1 - x)^2 \\
 &\in 100 * (\mathbf{y} - \mathbf{x}^2)^2 + (1 - \mathbf{x})^2 \\
 &= 100 * ([0.0, 0.3] - [0.7, 1]^2)^2 + (1 - [0.7, 1])^2 \\
 &= 100 * ([0.0, 0.3] - [0.49, 1])^2 + [0.0, 0.3]^2 \\
 &= 100 * [-1.0, -0.19]^2 + [0.0, 0.09] \\
 &= [3.61, 100.09].
 \end{aligned}$$

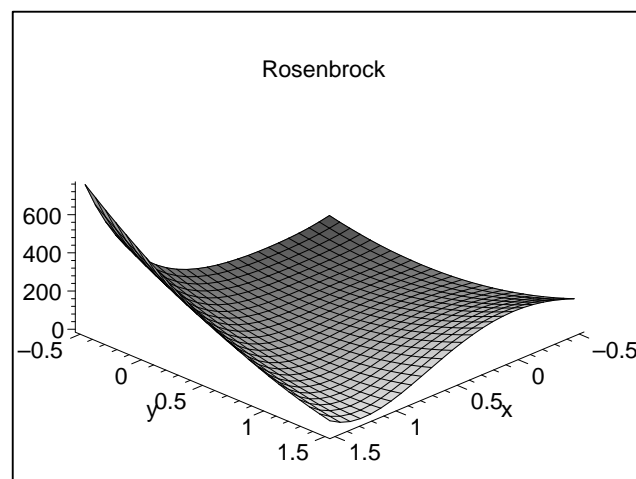
Guaranteed upper, lower bounds for range over $[0.7, 1] \times [0.0, 0.3]$

$f(0, 0) = 1$ box $[0.7, 1] \times [0.0, 0.3]$ cannot contain a global minimum

In a branch-and-bound global optimization algorithm, allows us to discard large regions where the answer is **not**

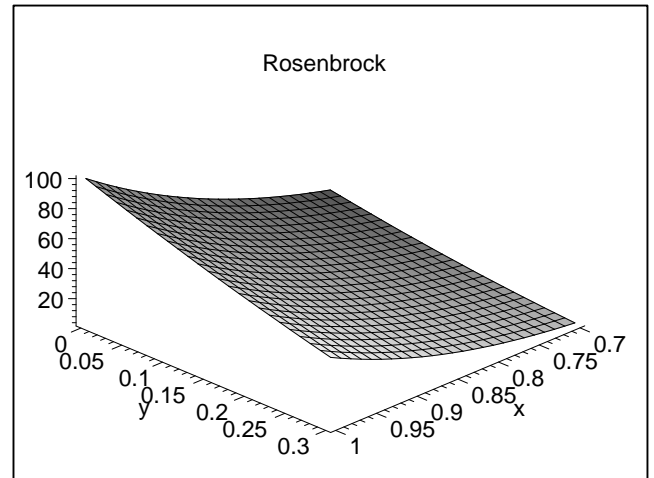
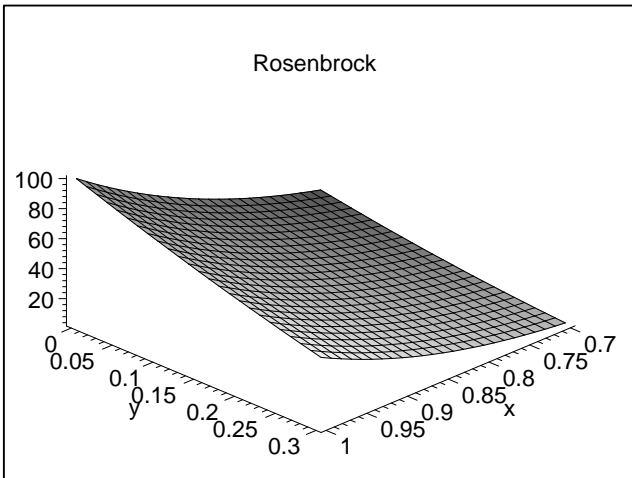
Interval Arithmetic - Exact Arithmetic

Rosenbrock function $f(x, y) := 100 * (y - x^2)^2 + (1 - x)^2$



Interval Arithmetic - Exact Arithmetic

Rosenbrock function $f(x, y) := 100 * (y - x^2)^2 + (1 - x)^2$



Example: ex3_rosenbrock.f95

Naive evaluation in interval arithmetic gives containment

```

MODULE M
  INTERFACE Rosenbrock
    MODULE PROCEDURE S1
    MODULE PROCEDURE S2
  END INTERFACE
CONTAINS
  REAL FUNCTION S1 (X, Y)
    REAL, INTENT(IN) :: X, Y
    S1 = 100 * (Y - X**2)**2 + (1 - X)**2
  END FUNCTION S1
  INTERVAL FUNCTION S2 (X, Y)
    INTERVAL, INTENT(IN) :: X, Y
    S2 = 100 * (Y - X**2)**2 + (1 - X)**2
  END FUNCTION S2
END MODULE M

PROGRAM EX3_ROSENBRUCK
  USE M
  INTERVAL :: X = [0.7, 1.0], Y = [0.0, 0.3]
  PRINT *, "interval X = ", X
  PRINT *, "interval Y = ", Y

  PRINT *, "bounds : ", Rosenbrock (X, Y)
  PRINT *, "minimum: ", Rosenbrock (0.7, 0.3)
  PRINT *, "maximum: ", Rosenbrock (1.0, 0.0)
END PROGRAM EX3_ROSENBRUCK

```

Result:

```

$ ex3
interval X = [0.69999999999999995,1.0]
interval Y = [0.0E+0,0.300000000000000005]
bounds : [3.60999999999999927,100.090000000000001]
minimum: 3.6999988
maximum: 100.0

```

Take-home: Computing guaranteed bounds is easy

Example: ex3_rosenbrock.m

Repeating in Matlab, ...

Naive evaluation in interval arithmetic gives containment

```

function z = Rosenbrock (x, y)
%ROSENBRUCK Evaluate the Rosenbrock function.
  z = 100 *(y - x^2)^2 + (1 - x)^2;

```

% File: ex3_rosenbrock.m

```

intvalinit('DisplayInfSup');
x = intval( '[0.7, 1.0]' )
y = intval( '[0.0, 0.3]' )

```

```

disp(' '); disp('bounds:');
z = Rosenbrock (x, y)
minimum = Rosenbrock (0.7, 0.3)
maximum = Rosenbrock (1.0, 0.0)

```

Result (lightly edited):

```

>> ex3_rosenbrock
intval x = [ 0.6999, 1.0000]
intval y = [ 0.0000, 0.3001]

bounds:
intval z = [ 3.6099, 100.0901]
minimum = 3.7000
maximum = 100

```

Interval Arithmetic - Exact Arithmetic

Exp: $\text{Exp.}, \exp([3, 4]) = [?, ?]$

$\exp(\underline{x}) = \exp([\underline{x}, \bar{x}]) = [\exp(\underline{x}), \exp(\bar{x})]$

Sqrt: Ex., $\sqrt{[3, 5]} = [?, ?]$

$\sqrt{[-0.001, 1]} = [?, ?]$

$\sqrt{[-2, -1]} = [?, ?]$

Reference: Bill Walster on exception-free interval arithmetic in Sun's Fortran compiler

Reference: John D. Pryce, *An Introduction to Containment Sets*

More general definition of operations:

$$x \circ y := \left\{ \tilde{x} \circ \tilde{y} : \tilde{x} \in x \cap \text{domain}(\circ), \tilde{y} \in y \cap \text{domain}(\circ) \right\}$$

Implies need for empty set \emptyset

$$\sqrt{x} = \emptyset \text{ or } [\sqrt{\max(0, \underline{x})}, \sqrt{\bar{y}}]$$

Elementary functions are monotonic?

sin, cos:

Argument reduction + monotonic

Example: ex4_functions.f95

Some elementary functions

```
PROGRAM EX4_FUNCTIONS
  INTERVAL :: A = [3, 5], B = [-0.001, 1], C = [-2, -1]
  PRINT *, "interval A = ", A
  PRINT *, "interval B = ", B
  PRINT *, "interval C = ", C
  PRINT *, ""

  PRINT *, "exp(A : ", exp(A)
  PRINT *, "sin(A) : ", sin(A)
  PRINT *, "sqrt(A) : ", sqrt(A)
  PRINT *, "sqrt(B) : ", sqrt(B)
  PRINT *, "sqrt(C) : ", sqrt(C)
END PROGRAM EX4_FUNCTIONS
```

Result:

```
$ ex4
interval A = [3.0,5.0]
interval B = [-1.0000000000000001E-3,1.0]
interval C = [-2.0,-1.0]

exp(A : [20.085536923187664,148.41315910257663]
sin(A) : [-1.0,0.14112000805986725]
sqrt(A): [1.7320508075688771,2.2360679774997899]
sqrt(B): [0.0E+0,1.0]
sqrt(C): [EMPTY]
```

Example: ex4_functions.m

Repeating in Matlab, ...
Some elementary functions

```
intvalinit('DisplayInfSup');
a = intval( '[3, 5]' )
b = intval( '[-0.001, 1]' )
c = intval( '[-2, -1]' )

disp ( ' ' ); disp ('exp:'); x = exp(a)
disp ('sin:'); x = sin(a)
disp ('sqrt:'); x = sqrt(a)
disp ('sqrt:'); x = sqrt(b)
disp ('sqrt:'); x = sqrt(c)
```

Result (lightly edited):

```
>> ex4_functions
intval a = [ 3.0000, 5.0000]
intval b = [ -0.0011, 1.0000]
intval c = [ -2.0000, -1.0000]

exp: intval x = [ 20.0855, 148.4132]
sin: intval x = [ -1.0000, 0.1412]
sqrt: intval x = [ 1.7320, 2.2361]
```

Result (continued):

```
sqrt:
Warning: SQRT: Real interval input out of range changed to be
> In ... \intl\intval\@intval\sqrt.m at line 122
In ... \matlab\ex4_functions.m at line 18

Warning: Complex Sqrt: Input interval intersects with branch
> In ... \intl\intval\@intval\sqrt.m at line 107
In ... \intl\intval\@intval\sqrt.m at line 124
In ... \matlab\ex4_functions.m at line 18

intval x = [ -0.0008 - 0.7075i, 1.4143 + 0.7075i]

sqrt:
Warning: SQRT: Real interval input out of range changed to be
> In ... \intl\intval\@intval\sqrt.m at line 122
In ... \matlab\ex4_functions.m at line 19

Warning: Complex Sqrt: Input interval intersects with branch
> In ... \intl\intval\@intval\sqrt.m at line 107
In ... \intl\intval\@intval\sqrt.m at line 124
In ... \matlab\ex4_functions.m at line 19

intval x = [ -0.2248 + 0.9999i, 0.2248 + 1.4495i]
```

Lab Exercise 4. Domains and Csets

Exercise 4.1. In Matlab, run ex4_functions.m:

```
>> ex4_functions
```

Exercise 4.2. (Optional) Make a copy of `ex4_functions.m`. Edit the copy to try different “interesting” interval arguments.

Exercise 4.3. (Optional) Make a copy of `ex4_functions.m`. Edit the copy to try different “interesting” functions.

Take-home: It matters how we define domains.

Interval Arithmetic - Rounded

Pattern: Interval arithmetic captures modeling, round-off, and truncation errors

But ...

Computer arithmetic is NOT exact (sigh ...)

Implementation uses IEEE 754 outwardly directed rounding

Add:

$$\mathbf{x} + \mathbf{y} = [\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\text{RoundDown}(\underline{x} + \underline{y}), \text{RoundUp}(\bar{x} + \bar{y})]$$

Others similarly

Elementary functions similarly (?)

Interval Arithmetic - Rounded

Tricks?

SetRound on some machines is/was slow

Batch or vectorize rounded operations

Represent \underline{x} as $-\bar{x}$, and make (almost) all roundings toward $+\infty$

With that information, you could all write a C++ class Interval of decent quality in a week

DON'T, except to learn something

“Der Teufel steckt im Detail”

“The Devil is in the details”

Production quality packages include:

- INTLAB - INTerval LABoratory from TU Hamburg-Harburg, www.ti3.tu-harburg.de/english/index.html
- PROFIL/BIAS for Matlab from TU Hamburg-Harburg, www.ti3.tu-harburg.de/Software/PROFILEnglisch.html
- Fortran 95 and C++ from Sun Microsystems, www.sun.com/software/sundev/suncc/index.html
- C-XSC, Pascal-XSC packages from TU Karlsruhe, www.uni-karlsruhe.de/~iam/html/language/xsc-sprachen.html
- FILIB and FILIB++, Hofschuster, Krämer, et al., Bergische Universität Wuppertal, www.math.uni-wuppertal.de/wrswt/software/filib.html

- Maple has range arithmetic, www.maplesoft.com
- Mathematica has RealInterval arithmetic, www.wri.com
- COSY from Michigan State University, cosy.pa.msu.edu

See also *Languages for Interval Analysis*, www.cs.utep.edu/interval-comp/intlang.html

Interval Arithmetic - I/O

Pattern: Interval I/O is subtle/interesting/hard

```
Interval A, B;  
A = 0.1;  
B = sin(1.0E50);
```

0.1 is infected by error on input to compiler or run-time

Is it a point?

It is converted to an interval?

Until recently, probably most hotly debated issue in interval community

Ada-like camp: (We know what's best for you)

Require explicit type conversions

Often from character strings

```
Interval A = Interval ("0.1");
```

C-like camp: (You **do** know what you are doing, nicht?)

Compiler should do what you said, even if it is not what you meant

Answer: Both?

Read the manual for the system you use

Carefully consider every input and every output

You **will** be surprised!

Example: `ex5_ce1_6.f95`

Reference: Sun Fortran 95 Interval Arithmetic Programming Reference, July 2001, Example 1-6, p. 22. Interval input/output

```
PROGRAM EX5_I0  
  INTERVAL :: X  
  INTEGER :: IOS = 0  
  CHARACTER*30 BUFFER  
  PRINT *, "Enter an interval, e.g., [3.14, 3.15], [0.1], 1.7  
  PRINT *, "Press Control/D to terminate!"  
  WRITE(*, 1, ADVANCE='NO')  
  READ(*, '(A12)', IOSTAT=IOS) BUFFER  
  DO WHILE (IOS >= 0)  
    PRINT *, ' Your input was: ', BUFFER  
    READ(BUFFER, '(Y12.16)') X  
    PRINT *, "Resulting stored interval is:", X
```

```

PRINT '(A, Y20.5)', ' Single number interval output (Y) interval('0.1e1')
PRINT '(A, VF30.9)', ' VF format interval output is: radius = rad(c)
WRITE(*, 1, ADVANCE='NO')                                c = intval('3.14_')
READ(*, '(A12)', IOSTAT=IOS) BUFFER                      radius = rad(c)
END DO
1 FORMAT(" X = ? ")
END PROGRAM EX5_IO

```

Result (lightly edited):

```

>> ex5_io
disp_(a):   intval a =      4.____
infsup(a):  intval a = [   3.0000,   4.0000]
midrad(a):  intval a = <   3.5000,   0.5000>

intval c = [   0.0999,   0.1001]
radius = 1.3878e-017
intval c = [   1.0000,   1.0000]
radius = 0
intval c = [   3.1299,   3.1501]
radius = 0.0100

```

Result:

```

$ ex5
Enter an interval, e.g., [3.14, 3.15], [0.1], 1.732, 1.7320000
Press Control/D to terminate!
X = ? [3.14, 3.15]
Your input was: [3.14, 3.15]
Resulting stored interval is:
[3.1399999999999996,3.1500000000000004]
Single number interval output (Y) is:      3.1
VF format interval output is: [ 3.139999999, 3.150000004]
X = ? [0.1]
Your input was: [0.1]
Resulting stored interval is:
[0.09999999999999991,0.10000000000000001]
Single number interval output (Y) is:      0.1000000000
VF format interval output is: [ 0.099999999, 0.100000004]
X = ? 0.1
Your input was: 0.1
Resulting stored interval is:
[0.0E+0,0.20000000000000002]
Single number interval output (Y) is: [0.0 ,.21 1]
VF format interval output is: [ 0.000000000, 0.200000004]
X = ? 0.10000
Your input was: 0.10000
Resulting stored interval is:
[0.09998999999999995,0.10001000000000001]
Single number interval output (Y) is:      0.1000
VF format interval output is: [ 0.099989999, 0.100010001]
X = ? 1.732
Your input was: 1.732
Resulting stored interval is:
[1.7309999999999998,1.7330000000000001]
Single number interval output (Y) is:      1.73
VF format interval output is: [ 1.730999999, 1.733000001]

```

Lab Exercise 5. Interval I/O

Exercise 5.1. In Matlab, run `ex5_io.m`:

```
>> ex5_io
```

Explain what you see.

Exercise 5.2. (Optional) Make a copy of `ex5_io.m`. Edit the copy to try different “interesting” interval arguments.

Exercise 5.3. (Optional) Try some of the commands

```

>> intvalinit('Display_')
>> intvalinit('DisplayMidrad')
>> format long
>> format short e

```

and execute `ex5_io.m`. Explain what you see.

Take-home: I/O matters.

Interval Arithmetic - I/O

See Appendix A: `ex6_bpv.f95` and Appendix B: `ex6_bpv.m`
Reference: I. Babuska, M. Prager, and E. Vitasek,
Numerical Processes in Differential Equations, Interscience,
1966.

Illustrates: Accumulation of rounding error

$$X_n = \int_0^1 x^n \exp(x-1) dx$$

$$X_n = 1 - nX_{n-1}$$

| N | X | Interval X |
|---|----------------|-------------|
| 1 | 0.367879450321 | 0.367879441 |
| 2 | 0.264241099358 | 0.264241118 |
| 3 | 0.207276701927 | 0.20727664 |
| 4 | 0.170893192291 | 0.17089341 |

Example: `ex5_io.m`

Interval input/output

```

a = intval( '[3, 4]' );

disp(' '); disp('disp_(a):'); disp_(a)
disp('infsup(a):'); infsup(a)
disp('midrad(a):'); midrad(a)

intvalinit('DisplayInfSup');
c = intval('0.1')
radius = rad(c)

```

| | | |
|----|-------------------|------------------------|
| 5 | 0.145534038544 | 0.1455329 |
| 6 | 0.126795768738 | 0.1268024 |
| 7 | 0.112429618835 | 0.112383 |
| 8 | 0.100563049316 | 0.10093 |
| 9 | 0.094932556152 | 0.0916 |
| 10 | 0.050674438477 | 0.084 |
| 11 | 0.442581176758 | 0.07 |
| 12 | -4.310974121094 | [.58094E-001, .15390] |
| 13 | 57.042663574219 | [-1.001, 0.2448] |
| 14 | -797.597290039062 | [-2.427, 15.01] |

Note: Table if from Fortran program in Appendix A.
 $X = \text{mid}(\text{ivl}X)$, but X is REAL, while Interval $\text{ivl}X$ is double

Linear systems $Ax = b$

Pattern: Good floating-point algorithms rarely make good interval algorithms

If A and/or b are interval-valued, solution is star-shaped
 Theorem (Rohn). Tight enclosure is an NP-hard problem in the dimension

Gaussian elimination?

Can give very over-estimated enclosures
 Often breaks with pivot containing zero
 Even for reasonable conditioned problems
 Pivoting, scaling, preconditioning help, but do not cure

Why? Dependency

For $n = 2$, symbolic expression for x_1 involves $a_{1,1}$ three times
 Seven times for $n = 3$
 Definitely **not** SUE!

Lesson: Don't use interval Gaussian elimination

See Appendix C. Sample interval Gaussian elimination code from Sun's F95 distribution

Linear systems $Ax = b$

If A and/or b are interval-valued, solution is star-shaped
 Neumaier example 2.7.3. The linear system of interval equations with

$$\mathbf{A} = \begin{pmatrix} [2, 2] & [-2, 1] \\ [-1, 2] & [2, 4] \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} [-2, 2] \\ [-2, 2] \end{pmatrix}$$

has the solution set shown. The hull of the solution set is $([-4, 4], [-4, 4])^T$, drawn with dashed lines. To construct the exact solution set is a nontrivial task, even for 2×2 systems!

Show Neumaier Figure 2.3

Linear systems $Ax = b$

Pattern: Contractive mapping theorems are the basis for effective computational algorithms for validating existence and uniqueness of solutions

If not Gaussian Elimination, then what?

Residual correction

Motivation:

Let x^* = Approximate solution of $Ax = b$

$$Ax = b$$

$$Ax - Ax^* = b - Ax^*$$

$$A(x - x^*) = Ae = r = b - Ax^*$$

(Benefits from accurate dot product, Karlsruhe)

Let e^* = Approximate solution of $Ae = r$

$x^{**} = x^* + e^*$ is a better approximate answer to $Ax = b$

Repeat

Show Neumaier example 2.6.3

Krawczyk's method $Ax = b$

Pattern: Contractive mapping theorems are the basis for effective computational algorithms for validating existence and uniqueness of solutions

Pattern: Know what is interval, what is point

Pattern: Compute two enclosures, intersect them

Neumaier, *Introduction to Numerical Analysis*, p. 116

Preconditioner C = Approximate inverse of midpoint (\mathbf{A})

For $\tilde{A} \in \mathbf{A}$ and $\tilde{b} \in \mathbf{b}$

Let \tilde{x}^* = true solution of $\tilde{A}\tilde{x}^* = \tilde{b}$

$$\tilde{x}^* = C\tilde{b} + (I - C\tilde{A})\tilde{x}^*$$

If interval \mathbf{x}^j with $\tilde{x}^* \in \mathbf{x}^j$, then $\tilde{x}^* \in C\mathbf{b} + (I - C\mathbf{A})\mathbf{x}^j$

Improve inclusion with Krawczyk operator:

$$\tilde{x}^* \in \mathbf{x}^{j+1} := (C\mathbf{b} + (I - C\mathbf{A})\mathbf{x}^j) \cap \mathbf{x}^j$$

Stop iterating when sum σ_j of radii of components no longer decreases rapidly

Krawczyk's method for $Ax = b$

Neumaier, *Introduction to Numerical Analysis*, p. 116

Krawczyk avoids inverting an interval matrix

$ier = 0$;

C = Approximate inverse of midpoint (\mathbf{A});

$\mathbf{a} = C * \mathbf{b}$;

$\mathbf{E} = I - C * \mathbf{A}$;

$\beta = \max_i \sum_k |\mathbf{E}_{i,k}|$; % Must be rounded upwards

if $(\beta \geq 1)$ { $ier = 1$; return; }

$\alpha = \|\mathbf{a}\|_\infty / (1 - \beta)$;

$\mathbf{x} = ([-\alpha, \alpha], \dots, [-\alpha, \alpha])^T$;

$\sigma_{old} = \text{inf}$; $\sigma = \sum_k \text{radius}(\mathbf{x}_k)$; $fac = (1 + \beta)/2$;

```

while ( $\sigma < fac * \sigma_{old}$ ) {
     $\mathbf{x} = (\mathbf{a} + \mathbf{E}\mathbf{x}) \cap \mathbf{x}$ ;
     $\sigma_{old} = \sigma$ ;  $\sigma = \sum_k \text{radius}(\mathbf{x}_k)$ ;
}

```

See Neumaier's examples

See Appendix D. Hybrid dense linear system solver from TU Hamburg-Harburg INTLAB for MATLAB distribution

Krawczyk's method for $Ax = b$

Neumaier, *Introduction to Numerical Analysis*, examples Show Examples 2.6.3

Lab Exercise 6. Linear Systems

Examples from Arnold Neumaier, *Introduction to Numerical Analysis*, Cambridge, 2001

Exercise 6.1. Copy and run the file `lab6_linear_a.m` to solve the system

$$A = \begin{pmatrix} 372 & 241 & -613 \\ -573 & 63 & 511 \\ 377 & -484 & 107 \end{pmatrix}, b = \begin{pmatrix} 210 \\ -281 \\ 170 \end{pmatrix}$$

Explain what you see.

Exercise 6.2. Scale the system by 10^{-3} and repeat. Hint: scaling by the point `* 1.0E-3` is different from scaling by the interval `* intval('[-1, 1]')` `* 1.0E-3`. Which is correct?

Exercise 6.3. (Optional) Repeat for the system

$$A = \begin{pmatrix} 372 & 241 & -125 \\ -573 & 63 & 182 \\ 377 & -484 & 437 \end{pmatrix}, b = \begin{pmatrix} 155 \\ -946 \\ 38 \end{pmatrix}$$

Exercise 6.4. (Optional) Repeat for the interval system

$$A = \begin{pmatrix} [2, 2] & [-2, 1] \\ [-1, 2] & [2, 4] \end{pmatrix}, b = \begin{pmatrix} [-2, 2] \\ [-2, 2] \end{pmatrix}$$

Can you characterize the exact solution? Hint: Consider the 2^5 "corners." Cheat by using `lab6_linear_d.m`

Exercise 6.5. (Optional) Repeat for your favorite linear systems. Try Hilbert, Vandermonde, etc.

Take-home: We can find enclosures for solutions of linear systems.

Sign-Accord method for $Ax = b$

See Rohn 1989; Madsen and Toft 1994

Solve a system of non-linear equations involving the "corners" of the linear system

O. Caprani, K. Madsen, and H. Nielsen, *Introduction to Interval Analysis*, www.imm.dtu.dk/courses/02611/IA.pdf

Nonlinear root finding $f(x) = 0$

Pattern: Contractive mapping theorems are the basis for effective computational algorithms for validating existence and uniqueness of solutions

Pattern: Know what is interval, what is point

Pattern: Compute two enclosures, intersect them

Fixed point: $T(x) = x$

For validation, **Banach Fixed-Point Theorem:**

Let $T : X \rightarrow X$ be continuous and defined on a complete metric space X with metric ρ . Let α satisfy $0 < \alpha < 1$, and let

$$\rho(T(x), T(y)) \leq \alpha \rho(x, y)$$

for all x and $y \in X$.

Then T has a unique fixed point $x^* \in X$.

or Schauder or Brower fixed point theorems

Interval Newton method $f(x) = 0$

Univariate interval Newton operator:

$$N(f; \mathbf{x}, \tilde{x}) := \tilde{x} - f([\tilde{x}]) / \mathbf{f}'(\mathbf{x})$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} \cap N(f; \mathbf{x}^{(k)}, \tilde{x}^{(k)})$$

N is a continuous map

$$\mathbf{x}^{(k+1)} \subset \text{Interior}(\mathbf{x}^{(k)})$$

validates contraction

Where does the box \mathbf{X} in which validation is attempted come from?

- Approximate root \tilde{x}
- Verify **existence** in as **small** a box as possible
- Verify **uniqueness** in as **large** a box as possible

Example: `ex7_simple_newton.m`

TU Hamburg-Harburg INTLAB for MATLAB Simple 1-D interval Newton stripped down from `verifynlss.m` (see Appendix E. `verifynlss.m`)

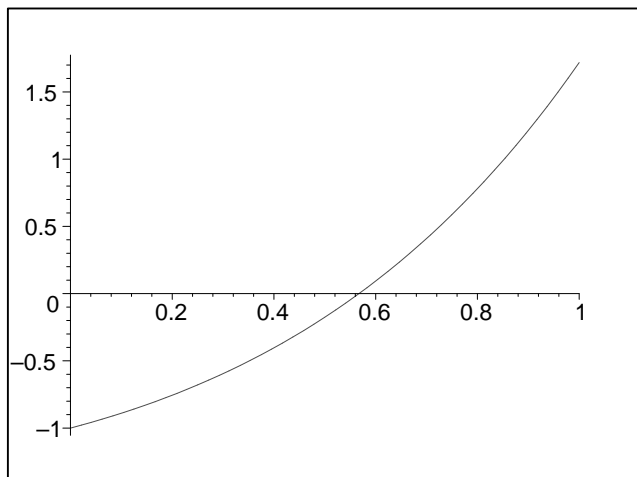
```
function X = SimpleIntervalNewton
    (f, trial_inclusion, tolerance)
    strfun = fcnchk(f);
    X = trial_inclusion;
    k = 0; kmax = 10;
    while (diam(X) > tolerance) & (k < kmax) & (~isnan(X))
        k = k + 1
        pt_x = intval(mid(X))
        pt_y = feval(strfun, pt_x)
        Y = feval(strfun, gradientinit(X))
        Newt = pt_x - pt_y / Y.dx
        isValidated = in0(Newt, X)
        X = intersect (X, Newt)
    end
```

% File: ex7_simple_newton.m

```
intvalinit('DisplayInfSup')
funct = 'x*exp(x)-1'
x = intval(' [0.7, 0.8] ')
x = SimpleIntervalNewton(funct, x, 1.0e-4)
disp(' ')
x = intval(' [0, 1] ')
x = SimpleIntervalNewton(funct, x, 1.0e-4)
disp(' ')
verifynlss (funct, 0.6, 'g', 1)
```

Where is the derivative? AD!

No root: $f(x) = x * \exp(x) - 1$

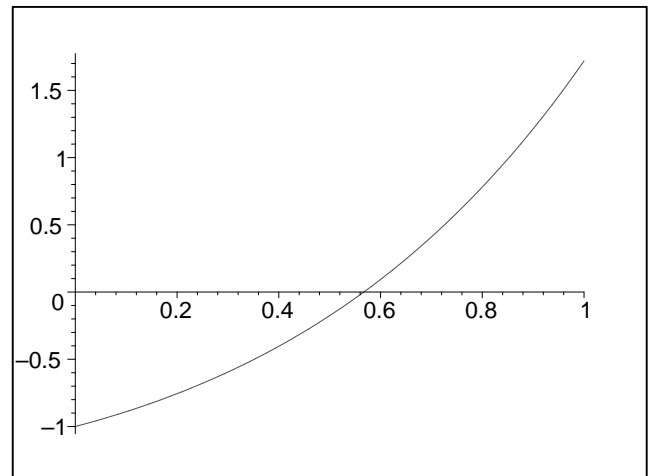


Initial interval: [0.7, 0.8]
 f(midpoint): 0.58
 Function: [0.4, 0.8]
 Gradient: [3, 4]
 Newton step from midpoint
 Interval Newton: [0.57, 0.61]

Intersect with [0.7, 0.8]: EMPTY

Take-home: No root. Guaranteed!

Root: $f(x) = x * \exp(x) - 1$



Initial interval: [0, 1]

f(midpoint): -0.17

Function: [-1, 1.8]

Gradient: [1, 5.5]

Newton step from midpoint

Interval Newton: [0.53, 0.68]

In interior of [0, 1]: Guaranteed root

Take-home: Root = [0.56713, 0.56715]. Guaranteed!

Result (lightly edited)

```
>> ex7_simple_newton
funct = x*exp(x)-1
intval x = [ 0.699999999999998, 0.800000000000001]
k = 1
pt_x = 0.750000000000000
pt_y = 0.58775001245951
intval gradient value Y.x =
    [ 0.40962689522933, 0.78043274279398]
intval gradient derivative(s) Y.dx =
    [ 3.42337960269980, 4.00597367128645]
intval Newt =
    [ 0.57831293029964, 0.60328160874539]
isValidated = 0
intval X = [ NaN, NaN]
intval x = [ NaN, NaN]

intval x = [ 0.000000000000000, 1.000000000000000]
```

```

k = 1
pt_x = 0.5000000000000000
pt_y = -0.17563936464994
intval gradient value Y.x =
  [-1.000000000000000, 1.71828182845905]
intval gradient derivative(s) Y.dx =
  [1.000000000000000, 5.43656365691810]
intval Newton =
  [0.53230705565756, 0.67563936464994]
isValidated = 1
intval X = [0.53230705565756, 0.67563936464994]
k = 2
pt_x = 0.60397321015375
pt_y = 0.10489220046736
intval gradient value Y.x =
  [-0.09355754225597, 0.32782668613874]
intval gradient derivative(s) Y.dx =
  [2.60929882252449, 3.29311579688603]
intval Newton =
  [0.56377382802016, 0.57212124777619]
isValidated = 1
intval X = [0.56377382802016, 0.57212124777619]
k = 3
pt_x = 0.56794753789818
pt_y = 0.00222377945412
intval gradient value Y.x =
  [-0.00928492047727, 0.01381141777554]
intval gradient derivative(s) Y.dx =
  [2.74800679879583, 2.78583338290301]
intval Newton =
  [0.56713830429107, 0.56714929222141]
isValidated = 1
intval X = [0.56713830429107, 0.56714929222141]
intval x = [0.56713830429107, 0.56714929222141]

```

```

residual norm(f(xs_k)), floating point iteration 1
ans = 0.09327128023431
residual norm(f(xs_k)), floating point iteration 2
ans = 0.00238906529578
residual norm(f(xs_k)), floating point iteration 3
ans = 1.688379540176754e-006
residual norm(f(xs_k)), floating point iteration 4
ans = 8.446576771348191e-013
interval iteration 1
intval ans = [0.56714329040978, 0.56714329040979]

```

Interval Newton method $f(x) = 0$

Univariate:

$$N(f; x, \tilde{x}) := \tilde{x} - f(\tilde{x})/f'(\tilde{x})$$

$$x^{(k+1)} := x^{(k)} \cap N(f; x^{(k)}, \tilde{x}^{(k)})$$

Multi-dimensional version:
 $G := \nabla f$; \tilde{X} approximate root;

while true {

```

G := f'(X);
Bound solution set of G(x - \tilde{x}) = -f(\tilde{x})
  for x - \tilde{x} \subset X1;
if x^{(k+1)} \subset Interior(x^{(k)}) {
  Have validated existence and uniqueness;
  Iterate to convergence;
  Done;
} else {
  X := X \cap (X1 + \tilde{x});
  \tilde{x} := midpoint(X);
}
}

```

Newton operator N is a continuous map

$$x^{(k+1)} \subset \text{Interior}(x^{(k)})$$

validates contraction

Krawczyk operator is a generalization. Avoids inverting an interval matrix

See INTLAB's `verifnls.m` in Appendix E

Slopes are tighter, but not validated existence

Interval Newton method $f(x) = 0$

Pattern: Contractive mapping theorems are the basis for effective computational algorithms for validating existence and uniqueness of solutions

Pattern: Know what is interval, what is point

Pattern: Approximate, then validate

Pattern: Compute two enclosures, intersect

Constraint Propagation

Pattern: Compute two enclosures, intersect

Taken from Logic programming

Ref: Van Hentenryck et al., *Numerica: A Modeling Language for Global Optimization*

Example: Root of $f(x) = x^2 + x - 5 = 0$ on $[-4, 4]$
 Solve for $x = 5 - x^2 = 5 - [0, 16] = [-11, 5]$ vs. $[-4, 4]$
 Solve for $x = \pm\sqrt{5 - x} = \pm\sqrt{5 - [-4, 4]} = \pm\sqrt{[1, 9]} = [-3, -1]$ or $[1, 3]$
 Further iterations give

| | | | | |
|-------------|------------|-----------|----------|----|
| | [-3, | -1] | [1, | 3] |
| [-2.828428, | -2.449489] | [1.41421, | | 2] |
| [-2.797933, | -2.729375] | [1.73205, | 1.89362] | |
| [-2.792478, | -2.780175] | [1.76249, | 1.80775] | |
| [-2.791502, | -2.789296] | [1.78668, | 1.79931] | |
| [-2.791327, | -2.790931] | [1.78904, | 1.79258] | |
| [-2.791295, | -2.791223] | [1.79092, | 1.79192] | |

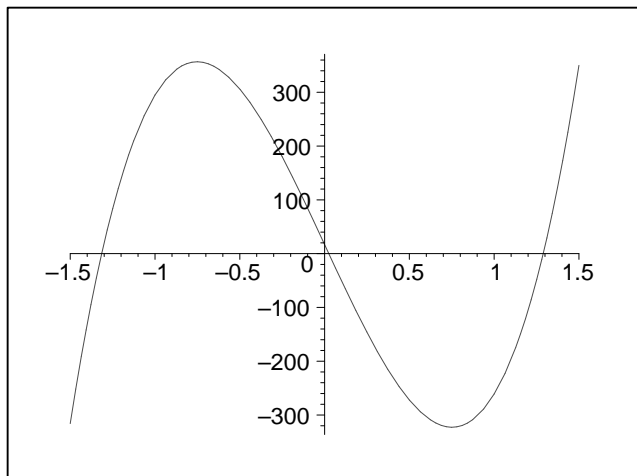
Convergence is approximately linear

No derivatives; no system to solve

Often reduces large boxes, while Newton reduces small boxes

Interval Newton is GLOBALLY Convergent

$$f(x) = -400(1.7 - x^2)x + 2x + 17$$



Univariate Newton operator:

$$N(f; \mathbf{x}, \tilde{x}) := \tilde{x} - f([\tilde{x}]) / f'(\mathbf{x})$$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} \cap N(f; \mathbf{x}^{(k)}, \tilde{x}^{(k)})$$

may divide by zero

Take-home: That is **not** a problem

Lab Exercise 7. Global Interval Newton

Example: $f(x) = -400(1.7 - x^2)x + 2x + 17$
on the interval $\mathbf{x} = [-0.5, 1.5]$, $\tilde{x} = \text{midpoint}(\mathbf{x})$

Exercise 7.1. Using pencil and paper, evaluate

1. $f(\tilde{x})$
2. $f'(\mathbf{x})$
3. $N(f; \mathbf{x}, \tilde{x}) := \tilde{x} - f([\tilde{x}]) / f'(\mathbf{x})$
4. $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} \cap N(f; \mathbf{x}^{(k)}, \tilde{x}^{(k)})$
Hint: You should get two intervals

Cheat by using `lab7_newton.m`

Exercise 7.2. (Optional) Repeat, using one of the intervals you got in 7.1.

Take-home: Divide by zero is not a problem.

Interval Newton is GLOBALLY Convergent

Example: $-400(1.7 - x^2)x + 2x + 17$

```
function X = OneINewton0 (f, trial_inclusion)
    strfun = fcnchk(f);
    X      = trial_inclusion;
    pt_x   = mid(X)
    pt_y   = feval(strfun, pt_x)
    Y      = feval(strfun, gradientinit(X));
    Y.dx   =
    del_X  = Extended_divide (intval(pt_y), Y.dx)
    new_X  = Extended_subtract (pt_x, del_X)
    X      = Extended_intersect (X, new_X);
```

test_OneINewton.m

% File: test_OneINewton.m

```
q = '-400*(1.7 - x^2)*x + 2*x + 17'
```

```
disp(' '); disp('No root in');
X = intval(' [2.1, 2.2] ');
X = OneINewton0 (q, X)
disp('No root');
disp(' '); disp('Press [Enter] for next test');
pause;

disp('One root in');
X = intval(' [1, 2] ');
X = OneINewton0 (q, X)
disp('Note: Tightened one end, but not the other');
disp('Press [Enter] for three more iterations');
pause;
X = OneINewton0 (q, X)
disp('Note: new_X is in the interior of old X ==> Exist unique');
X = OneINewton0 (q, X)
X = OneINewton0 (q, X)
disp('Note: Converging');
disp(' '); disp('Press [Enter] for next test');
pause;

disp('TWO roots in');
X = intval(' [-0.5, 1.5] ');
X = OneINewton0 (q, X)
disp('Note: Have resolved two roots');
```

Extended divide and subtract

```
function z = Extended_divide (a, b)
    if in(0, b)
        if 0 == inf(b)
            z = a * hull (intval(1)/sup(b), inf);
        elseif 0 == sup(b)
            z = a * hull (-inf, intval(1)/inf(b));
```

```

elseif in (0, a)
    z = [-inf, inf];
else
    z = [ (a * hull (intval(1)/sup(b), inf)),
          (a * hull (-inf, intval(1)/inf(b))) ];
end;
else
    z = a / b;
end;

```

```

function z = Extended_subtract (a, b)
if length(b) == 1
    z = a - b;
else
    z = [ a - b(1), a - b(2) ];
end;

```

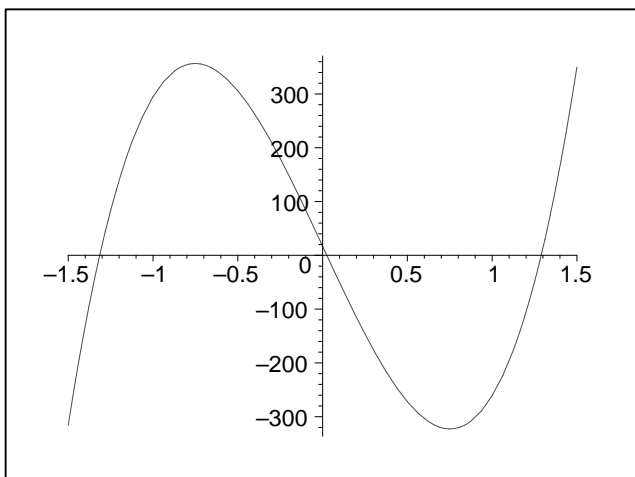
Extended intersect

```

function z = Extended_intersect (a, b)
if length(b) == 1
    z = intersect(a, b);
else
    a1 = intersect(a, b(1));
    a2 = intersect(a, b(2));
    if isempty(a1)
        z = a2;
    elseif isempty(a2)
        z = a1;
    else
        z = [a1, a2];
    end;
end;

```

$f(x) = -400(1.7 - x^2)x + 2x + 17$
No root in [2.1, 2.2]

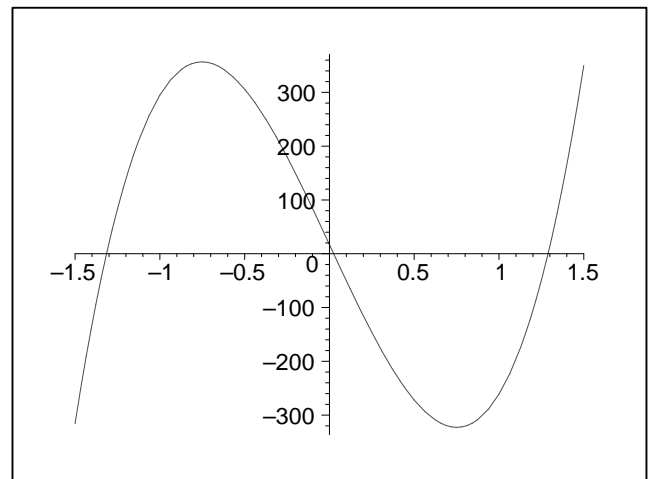


```

pt_x = 2.150000000000000
pt_y = 2.534650000000000e+003
intval ans = 1.0e+003 * [ 4.613999999999999, 5.130000000000000]
intval del_X = [ 0.49408382066276, 0.54933896835718]
intval new_X = [ 1.60066103164282, 1.65591617933724]
intval X = [ NaN, NaN]
No root

```

$f(x) = -400(1.7 - x^2)x + 2x + 17$
Root in [1, 2]



Result (lightly edited):

```

>> test_OneINewton
q = -400*(1.7 - x^2)*x + 2*x + 17

One root in
intval X = [ 1.000000000000000, 2.000000000000000]
pt_x = 1.500000000000000
pt_y = 3.500000000000000e+002
intval ans = 1.0e+003 * [ 0.522000000000000, 4.122000000000000]
intval del_X = [ 0.08491023774866, 0.67049808429119]
intval new_X = [ 0.82950191570881, 1.41508976225134]
intval X = [ 1.000000000000000, 1.41508976225134]
Note: Tightened one end, but not the other

Press [Enter] for three more iterations
pt_x = 1.20754488112567
pt_y = -97.39573068747973
intval ans = 1.0e+003 * [ 0.522000000000000, 1.7249748422742]
intval del_X = [-0.18658185955456, -0.05646211660633]
intval new_X = [ 1.26400699773200, 1.39412674068023]
intval X = [ 1.26400699773200, 1.39412674068023]
Note: new_X is in the interior of old X ==> Exist unique
pt_x = 1.32906686920611
pt_y = 54.96811396967802
intval ans = 1.0e+003 * [ 1.23925642837855, 1.6543072428956]
intval del_X = [ 0.03322727033067, 0.04435572227905]

```

```

intval new_X = [ 1.28471114692707, 1.29583959887545]
intval X =     [ 1.28471114692707, 1.29583959887545]
pt_x = 1.29027537290126
pt_y = 1.41891222048172
intval ans = 1.0e+003 * [ 1.30257927724640, 1.33704031921641]
intval del_X = [ 0.00106123368165, 0.00108930968370]
intval new_X = [ 1.28918606321756, 1.28921413921961]
intval X =     [ 1.28918606321756, 1.28921413921961]
Note: Converging

```

R.E. Moore, A Test for Existence of Solutions for Non-Linear Systems, *SIAM J. Numer. Anal.* 4, 611-615, 1977.

Exercise 8.1. `help verifynlss`

Exercise 8.2. Evaluate an inclusion of the zero of $x * \exp(x) - 1 = 0$ near $xs = 0.6$. Hint: `X = verifynlss('x*exp(x)-1', 0.6)`

Exercise 8.3. (Optional) Repeat, for roots of the example $f(x) = -400(1.7 - x^2)x + 2x + 17$ we have used above

To cheat: `lab8_verifynlss.m`

Take-home: Approximate, then validate.

Take-home: Define your own f to use in existing software.

Bounding ranges of functions

If $\tilde{x} \in X$, then $f(\tilde{x}) \in [?, ?]$

THE fundamental problem of interval arithmetic

- Naive interval evaluation
- Mean Value Theorem - Centered form
- Slope form
- Taylor series with remainder
- Taylor models (Berz, et al.)

Peter Hertling, "A Lower Bound for Range Enclosure in Interval Arithmetic," *Theor. Comp. Sci.*, 279:83-95, 2002.
www.informatik.fernuni-hagen.de/thi1/peter.hertling/publications.html

"This is a clearly written paper whose careful study should lay bare (and help to settle) questions about orders of convergence of over-estimation by interval computation of one kind or another as domain interval widths go to zero."
 – Ray Moore, April 25, 2002

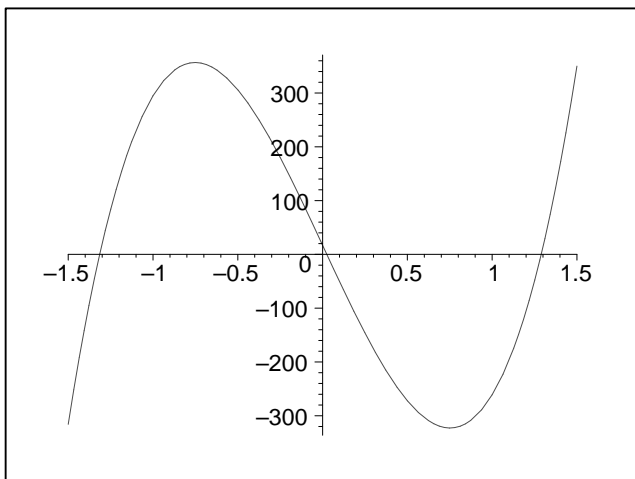
Bounding ranges — "Here be dragons!"

If $\tilde{x} \in X$, then $f(\tilde{x}) \in [?, ?]$

- Naive interval evaluation
- Mean Value Theorem - Centered form
- Slope form
- Taylor series with remainder
- Taylor models (Berz, et al.)

R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, Penn., 1979

Example: $-400(1.7 - x^2)x + 2x + 17$.
TWO roots in $[-0.5, 1.5]$



```

pt_x = 0.500000000000000
pt_y = -272
intval ans = 1.0e+003 * [-1.278000000000000, 2.022000000000000]
intval del_X = [ - Inf, -0.13452027695351]
               [ 0.21283255086071, Inf]
intval new_X = [ 0.63452027695351, Inf]
               [ - Inf, 0.28716744913929]
intval X =     [ 0.63452027695351, 1.500000000000001]
               [-0.500000000000001, 0.28716744913929]

```

See Appendix F. `Interval_Newton.m`, which uses a stack to find three roots for this example
 Run `test.Interval_Newton.m`

Lab Exercise 8. General root finding

INTLAB provides `verifynlss`, a nonlinear system solver based on the Krawczyk operator, see

R. Krawczyk, Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, *Computing* 4, 187-201, 1969.

G. Alefeld and J. Herzberger, *Einführung in die Intervallrechnung*, Springer-Verlag, Heidelberg, 1974
Introduction to Interval Computations, Academic Press, New York, 1983

H. Ratschek and J. Rokne, *Computer Methods for the Range of Functions*, Halsted Press, Wiley, New York, 1984

A. Neumaier, *Interval Methods for Systems of Equations*, Cambridge University Press, 1990

E. R. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, New York, 1992

Baker Kearfott, *Rigorous Global Search: Continuous Problems*, Kluwer, 1996

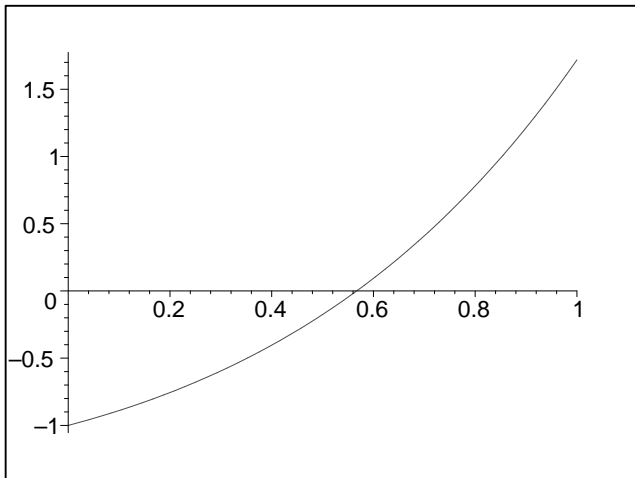
A. Neumaier, *Introduction to Numerical Analysis*, Cambridge University Press, 2001

Arnold Neumaier, Taylor forms - use and limits, can be downloaded from

www.mat.univie.ac.at/~neum/papers.html#taylor, May 2002

Many, many more

Naive, MVT: $f(x) = x * \exp(x) - 1$



On interval $\mathbf{x} = [0, 1]$

Naive interval evaluation: $[-1, 1.72]$

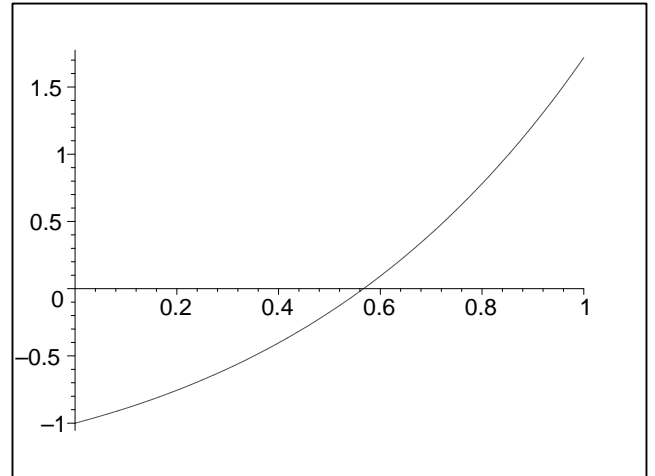
$$f(x) \in f(\mathbf{x})$$

Mean Value Theorem - Centered form: $[-2.9, 2.55]$

$$f(x) \in f([\tilde{x}]) + f'(x)(x - \tilde{x})$$

since $f'(x) \in [1, 5.44]$

Slope form: $f(x) = x * \exp(x) - 1$



On interval $\mathbf{x} = [0, 1]$

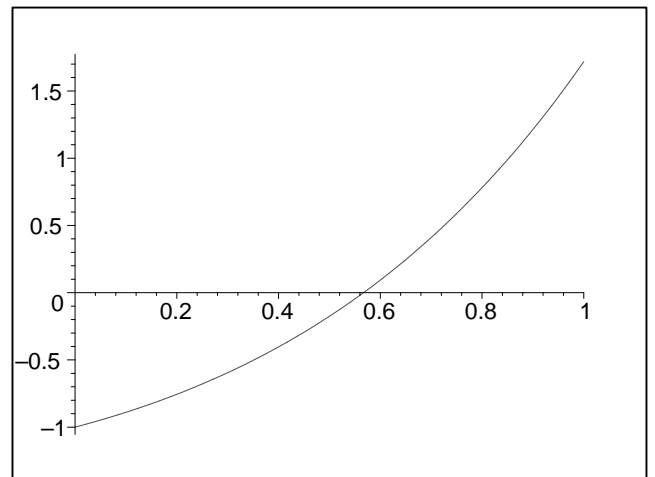
Slope form: $[-2.07, 1.72]$

$$f(x) \in f([\tilde{x}]) + \mathbf{S}(f, \mathbf{x}, \tilde{x})(x - \tilde{x}),$$

$\mathbf{S}(f, \mathbf{x}, \tilde{x})$ contains $\{(f(y) - f(\tilde{x})) / (y - \tilde{x}) : y \neq \tilde{x} \text{ in } \mathbf{x}\}$
 since $\mathbf{S}(f, \mathbf{x}, \tilde{x}) \in [1.64, 3.79]$

Multivariate extensions

Taylor series: $f(x) = x * \exp(x) - 1$



Taylor series with remainder (Moore, 1996):

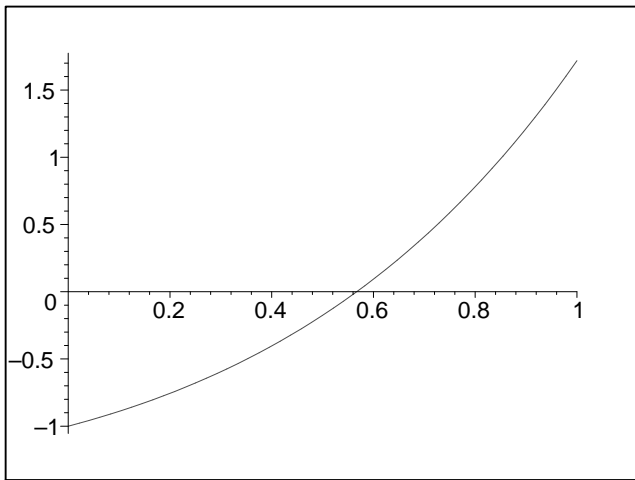
$$f(y) \in \left\{ \sum_{k=0}^p f^{(k)}([\tilde{x}]) (y - \tilde{x})^k / k! + R * (y - \tilde{x})^{p+1} : y \in \mathbf{x} \right\},$$

Remainder R contains $\{f^{(p+1)}(\xi) / (p+1)! : y, \xi \in \mathbf{x}\}$

Coefficients are narrow intervals
 Remainder is relatively thick
 Then bound the Taylor polynomial
 Enclosure looks like an Easter lily

Multivariate extensions

Taylor model: $f(x) = x * \exp(x) - 1$



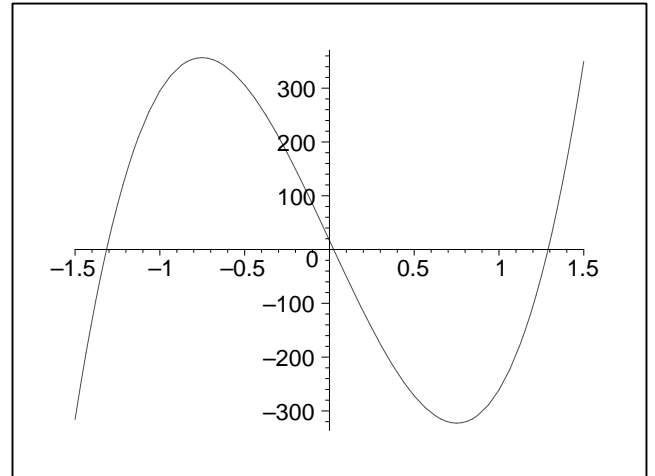
Taylor models (Krückelber, Kaucher, Miranker, Berz, ...):

$$f(y) \in \left\{ \sum_{k=0}^p T_k(y - \tilde{x})^k / k! + \mathbf{R} : y \in \mathbf{x} \right\}$$

Coefficients are **floating point numbers**
 Remainder \mathbf{R} is a relatively thick interval **constant**
 Then bound the Taylor polynomial
 Enclosure looks like a tube

Multivariate extensions

Lab Exercise 9. Bounding Ranges



For the example $f(x) = -400(1.7 - x^2)x + 2x + 17$ we have used above

Exercise 9.1. (Optional) Bound the range on the interval $\mathbf{x} = [1.2, 1.4]$ using: naive evaluation, Mean Value form, slopes, Taylor series, and Taylor models.

Exercise 9.2. (Optional) Repeat using the interval $\mathbf{x} = [0, 2]$

Exercise 9.3. (Optional) Repeat using the interval $\mathbf{x} = [1, a]$, as $a \rightarrow 1$

Take-home: Guaranteed bounds are easy. Tight bounds may be harder.

Global optimization

$f : X \subset \mathbb{R}^n \rightarrow \mathbb{R}$ continuous,
 differentiable objective function

$$\min_{x \in X} f(x),$$

possibly with linear or nonlinear constraints

Kearfott, *Rigorous Global Search: Continuous Problems*

Seek validated, tight bounds for minimizer X^* ,
 optimum value f^*

Challenges:

- Many local minima
- Flat objective
- High dimension

Moore-Skelbo Branch and Bound

Maintain three lists of boxes:

- \mathcal{L} not fully analyzed;
- \mathcal{R} small boxes validated to enclose a unique local minimum; and
- \mathcal{U} small boxes not validated either to enclose a minimum or *not* to enclose a minimum.

To find enclosures for f^* and X^* such that

$$f^* = f(X^*) \leq f(X),$$

for all $X \in \mathbf{X}_0 \subset \mathbb{R}^n$:

Discarding where the solution is NOT

- Range on a sub-box $>$ a known upper bound for f^*
- Non-zero gradient
- Hessian of f is not positive definite
- Fritz-John or Kuhn-Tucker conditions
- Feasibility of constraints

Branch and Bound

```

Initialize  $\mathcal{L} = \{\mathbf{X}_0\}$  // List of pending boxes
 $\bar{f} = +\infty$  //  $f^* \leq \bar{f}$ 
DO WHILE  $\mathcal{L}$  is not empty
  Get current box  $\mathbf{X}^{(c)}$  from  $\mathcal{L}$ 
  IF  $\bar{f} < f(\mathbf{X}^{(c)})$  THEN
    // We can guarantee there is
    no global minimum in  $\mathbf{X}^{(c)}$ 
    Discard  $\mathbf{X}^{(c)}$ 
  ELSE
    Attempt to improve  $\bar{f}$ 
  END IF
  IF we can otherwise guarantee that there
  is no global minimum in  $\mathbf{X}^{(c)}$  THEN
    Discard  $\mathbf{X}^{(c)}$ 
  ELSE IF we can validate the existence of
  a unique local minimum in  $\mathbf{X}^{(c)}$  THEN
    Shrink  $\mathbf{X}^{(c)}$ , and add it to  $\mathcal{R}$ 
  ELSE IF  $\mathbf{X}^{(c)}$  is small THEN
    Add  $\mathbf{X}^{(c)}$  to  $\mathcal{U}$ 
  ELSE
    Split  $\mathbf{X}^{(c)}$  into sub-boxes, and add them to  $\mathcal{L}$ 
  END IF
END DO

```

Quadrature

Suppose we have a rule

$$\int_a^b f(x) dx \in \text{Value}(f, a, b) + [\text{Remainder}(f, a, b)]$$

E.g., Lower and upper Riemann sums, Newton-Coates rules, Gaussian quadrature

Perform standard adaptive quadrature, but based on guaranteed error bounds

Ref: Gray and Rall; Corliss, Rall, and Krenz; Stork

Maintain list L : $[a_i, b_i, \text{Value}_i, \text{Remainder}_i]$ sorted by decreasing width(Remainder_i)

Self-Validating Adaptive Quadrature (SVLAQ):

Input:

Problem: f, a, b , Tolerance

Output:

Integral: $[c, d]$ or ERROR

Output assertion: If not ERROR

$\int_a^b f(x) dx \in [c, d]$ s.t. $d - c < \text{Tolerance}$

Initialize $L = [a, b, \text{Value}(f, a, b), [\text{Remainder}(f, a, b)]]$

while $\sum_L \text{width}(\text{Remainder}_i) < \text{Tolerance}$ do

$I = L.\text{pop}()$ % Subinterval with widest Remainder

$I_1, I_2 = L.\text{bisect}()$

$L.\text{insert}([I_1, \text{Value}(f, I_1), \text{Remainder}(f, I_1)])$

$L.\text{insert}([I_2, \text{Value}(f, I_2), \text{Remainder}(f, I_2)])$

Better stopping test

Adapt based on Remainders only

Adapt based on point estimates

Efficient reuse of computed values

Quadrature: Taylor Models

Recall Taylor models (Krückelber, Kaucher, Miranker, Berz, ...):

$$f(y) \in \sum_{k=0}^p T_k(y - \tilde{x})^k / k! + \mathbf{R}, \quad \forall y \in \mathbf{x}$$

Coefficients are floating point numbers

Remainder \mathbf{R} is a relatively thick interval constant

Then

$$\int f(y) \in \sum_{k=0}^p \frac{T_k}{(k+1)!} [(b - \tilde{x})^{k+1} - (a - \tilde{x})^{k+1}] + \mathbf{R} * [b - a]$$

and correspondingly for modest dimensions

Degree 6 in 6 dimensions is feasible

Careful of what is point and what is interval

Quadrature: Solve ODE?

$$g(x) = \int_a^x f(t) dt \text{ satisfies } g' = f(x), \quad g(a) = 0$$

Sure, but integration is easy, and ODE's are hard?

Try it. You might be surprised

Ordinary differential equations (ODE's)

Numerical solution for ODE's

$$u' = f(t, u), u(t_0) = u_0 \in [u_0]$$

Example – Lorenz system

$$\begin{aligned} x' &= \sigma(y - x) & x(0) &= 10 \\ y' &= rx - y - xz & y(0) &= 10 \\ z' &= xy - bz & z(0) &= 10 \\ \sigma &= 10, b = 8/3, r = 28 \end{aligned}$$

What is a “validated solution” of an ODE?

- Guarantee that there exists a unique solution u in the interval $[t_0, t_f]$
- Compute an interval-valued function $\mathbf{u}(t) = \hat{u}(t) + \mathbf{e}(t)$ such that

$$\begin{aligned} u(t, u_0) &\in \mathbf{u}(t), \text{ and} \\ w(\mathbf{u}(t)) &\leq \text{Tolerance}, t \in [t_0, t_f], u_0 \in \mathbf{u}_0, \end{aligned}$$

or equivalently,

- Compute an approximate solution $\hat{u}(t)$ and an interval-valued error function $\mathbf{e}(t)$ such that

$$\begin{aligned} u(t, u_0) &\in \hat{u}(t) + \mathbf{e}(t), \text{ and} \\ w(\mathbf{e}(t)) &\leq \text{Tolerance}, t \in [t_0, t_f], u_0 \in \mathbf{u}_0 \end{aligned}$$

ODE: Moore, Eijgenraam, Lohner, et al.

See more detail in Appendix G

AWA (Anfangwertaufgabe) by Rudolf Lohner (1978, 1987, and following)

loop for each integration step

Algorithm I: Validate existence and uniqueness

Algorithm II: Compute tighter enclosure

ODE: Lohner's Algorithm 1

Theorem [Walter1970a, Lohner1988a]. If we can find intervals $[t_0, t_0 + h]$ and $[u_0]_0$ such that

$$[u_0]_1 := u_0 + [0, h] * f([t_0, t_0 + h], [u_0]_0) \subseteq [u_0]_0,$$

then the ODE has a unique solution.

Algorithm I. Validate existence and uniqueness using Picard-Lindelöf iteration

Input:

Problem: $u' = f(t, u)$ Final time: t_f
 Initial conditions: $t_j, u(t_j) \in \mathbf{u}_j$ Tolerance

Output:

Coarse enclosure: \mathbf{u}_{j_0} Step size: h

Output assertions:

$u' = f(t, u), u(t_j) = u_j \in \mathbf{u}_j$ has a unique solution in $[t_j, t_j + h] \times \mathbf{u}_{j_0}$
 $u(t) \in \mathbf{u}_{j_0}$ for all $t \in [t_j, t_j + h]$

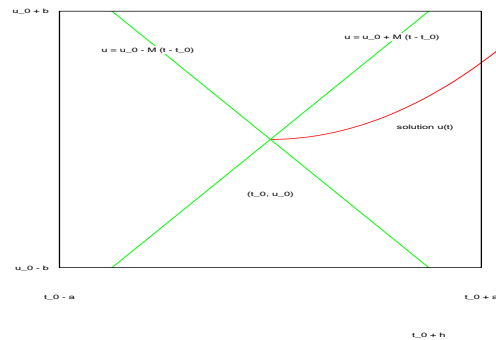
ODE: Existence and Uniqueness

If our computer programs can

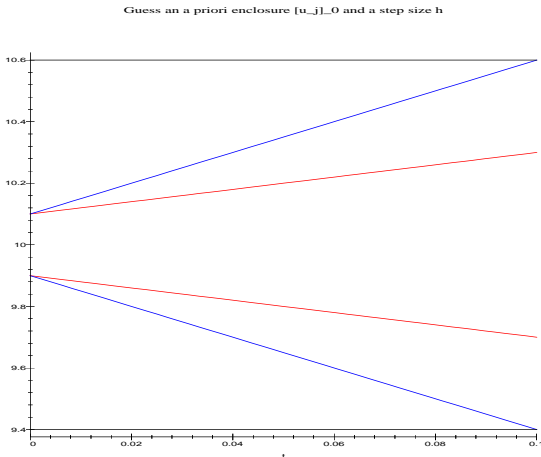
1. Find an interval $[t_0, t_1]$ and an interval $[u_0] = [u_0 - b, u_0 + b] \supseteq u_0$ for which
2. f is continuous in $D := [t_0, t_1] \times [u_0]$,
3. $\|f\| \leq M \in D$, and
4. $t_1 = \min(t_1, t_0 + b/M)$,

then we can guarantee that the ODE has a unique solution.

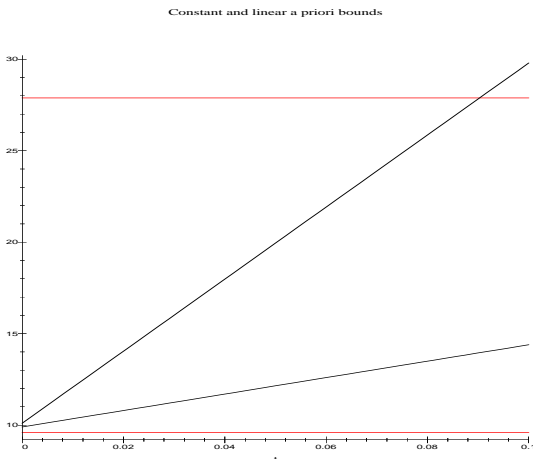
Existence and Uniqueness Theorem



ODE: Fig. 4. Guess *a priori* enclosure and step size # 1



ODE: Fig 8. Constant and linear *a priori* bounds # 2



ODE: Algorithm II. Compute tighter enclosure

Input:

- Problem: $u' = f(t, u)$
- Initial conditions: $t_j, u(t_j) \in \mathbf{u}_j$
- Current solution: matrices A_0, A_1, \dots, A_{j-1} and interval \mathbf{x}_j
- Coarse enclosure: \mathbf{u}_{j_0}
- Step size: h
- Tolerance: Tolerance

Input assertion:

$$u(t) \in \mathbf{u}_{j_0} \text{ for all } t \in [t_j, t_j + h]$$

Output:

Either

- Matrix: A_j
- Interval: \mathbf{x}_{j+1}
- Next node: $t_{j+1} \leq t_j + h$
- Solution: $\mathbf{u}_{j+1} := \text{Convex hull } (A_0 A_1 \dots A_j \mathbf{x}_{j+1})$

or else

Failure

Also available:

Approximation solution: $\hat{u}(t)$

Continuous enclosure on each subinterval: $\mathbf{u}_{j_1}(t)$

Output assertion:

$$u(t_{j+1}) \in A_0 A_1 \dots A_j \mathbf{x}_{j+1}$$

$$w(\mathbf{u}_{j+1}) \leq \text{Tolerance}$$

Basic idea:

Truncated Taylor series plus the remainder term

Local coordinate transformations

ODE: Naive Taylor Enclosure

Local Taylor expansion encloses u :

$$\begin{aligned} u(t) &\in \mathbf{u}_{j_1}(t) \\ &:= \mathbf{u}_j + \mathbf{u}'(t_j)(t - t_j) + \dots \\ &\quad + \frac{\mathbf{u}^{(p-1)}(t_j)}{(p-1)!}(t - t_j)^{p-1} + \frac{\mathbf{u}^{(p)}(T_j)}{p!}(t - t_j)^p \end{aligned}$$

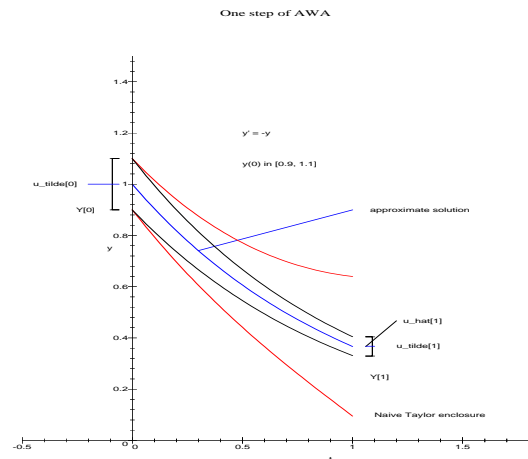
Derivatives are computed by automatic differentiation

Let $\mathbf{u}_{j+1} := \mathbf{u}_{j_1}(t_j + h)$

Gives width $(\mathbf{u}_{j+1}) > \text{width}(\mathbf{u}_j)$

Example: $y' = -y, y_0 \in [0.9, 1.1]$

ODE: Fig F. Naive Taylor enclosure does not contract



ODE: Algorithm II. Compute tighter enclosure

Point: Enclosures computed by naive Taylor enclosure do not contract, even when the true trajectories are strongly contracting

Need to be more clever.

[Eijgenraam 1981] and [Lohner 1978, 1987] express the enclosure as

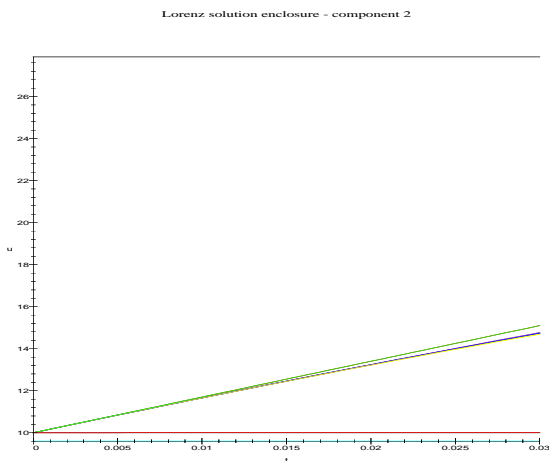
$$u(t) \in \left(\begin{array}{l} \text{point-valued truncated Taylor} \\ \text{series for approximate solution} \end{array} \right) + \left(\begin{array}{l} \text{old} \\ \text{error} \end{array} \right) + \left(\begin{array}{l} \text{enclosure of local} \\ \text{truncation error} \end{array} \right)$$

AWA follows contractive flows and controls the wrapping effect

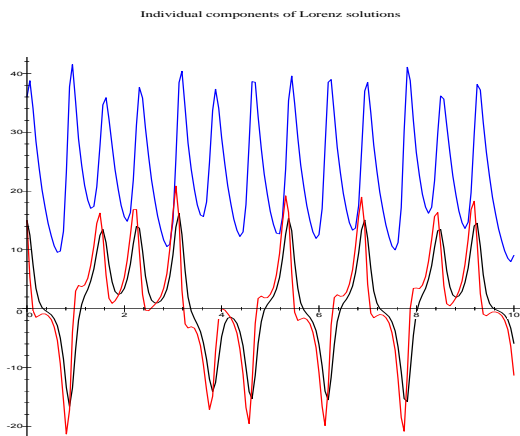
Algorithm II:

1. Point-valued approximate solution $\hat{u}_j(t)$;
2. Enclosure z_{j+1} of the local error of $\hat{u}_j(t)$;
3. Enclosure for the transformation matrix A_j ;
4. Prepare for next solution step;

ODE: Fig B. Enclosure of Lorenz component 2



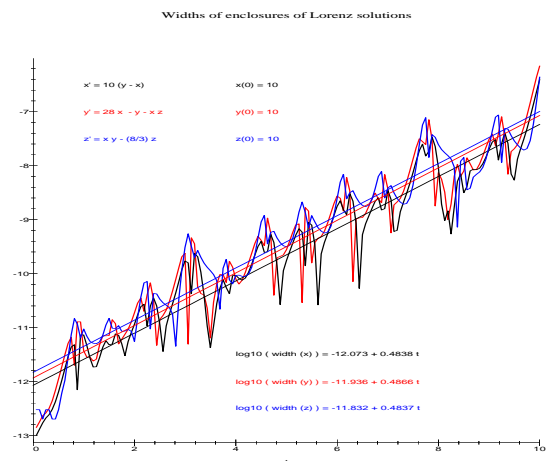
ODE: Fig H. Individual components of Lorenz solutions



ODE: Fig J. Lorenz attractor computed to $t = 10$ by AWA



ODE: Fig I. Widths of enclosures of Lorenz solutions



ODE: Advances on AWA

Martin Berz, et al., COSY INFINITY, cosy.pa.msu.edu/
Taylor models for beam physics and other problems

Ole Stauning, 1997 IMM DTU thesis *Automatic Validation of Numerical Solutions*
ADIODES (Automatic Differentiation Interval Ordinary Differential Equation Solver), www.imm.dtu.dk/documents/ftp/phdliste/phd36.abstract.html

Ned Nedialkov, 1999 U of Toronto thesis *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*
VNODE (Validated Numerical ODE), www.cas.mcmaster.ca/~nedialk/Software/VNODE/VNODE.shtml

Uses an interval Hermite-Obreschkoff method (modified implicit Taylor method)

Differential Algebraic Equations (DAE's)

$$F(x, x', t) = 0$$

Combination of differential and algebraic equations

Ref: Chang and Corliss; Pryce; Nedialkov

Simple pendulum example:

$$\begin{aligned}\frac{d^2x}{dt^2} &= -x\lambda(t) \\ \frac{d^2y}{dt^2} &= -y\lambda(t) - g \\ \text{condition} &= x * x + y * y - 1 = 0\end{aligned}$$

g is gravitational force

$\lambda(t)$ is the unknown tension in the string

Algebraic constraint: string does not stretch

Strategy: Generate Taylor series

Denote Taylor coefficient $x_i = x^{(i)}(0)(t-0)^i/i!$

loop for each integration time step

Find x_0 and y_0 : condition = 0

Find x_1 and y_1 : $d \text{condition} / dt = 0$

Solve $\frac{d^2x}{dt^2}, \frac{d^2y}{dt^2}, \frac{d^2 \text{condition}}{dt^2}$ for x_2, y_2, λ_0

Differentiate and solve for x_i, y_i, λ_{i-2}

Extend $x(t)$ and $y(t)$ by summing

Pryce: Determining dependencies and order is equivalent to LP transportation problem

Strategy works in general and can be automated

Nedialkov and Pryce put into VNODE

ODE: Some Open Problems

Quasimonotone problems: No readily available software, especially for 1-dimensional problems

Step size control: Several experimental step size control strategies have been attempted

Variable order: Several experimental order control strategies have been attempted

Global error "prediction": Kerbl explored a step size control based on using an approximate solution to predict regions where the trajectories are converging and where they are diverging

Representing the solution: Lohner has explored several representations of A_j , but there is no clearly best solution

Stiff problems: Very little is known about validated techniques for stiff problems. VNODE uses Hermite-Obreschkoff?

Applications: AWA, COSY, ADIODES, and VNODE have been used to solve real-world applications problems, but there are PLENTY more to do

Design Patterns in Validated Computing

Ref: GOF: Gamma, Helm, Johnson, and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, encapsulate the concept that varies

Corliss's design patterns in validated computing:

- Interval arithmetic captures modeling, round-off, and truncation errors
- Simplify first. Do arithmetic at last moment. Use SUE
- Interval I/O is subtle/interesting/hard
- Good floating-point algorithms rarely make good interval algorithms
- Contractive mapping theorems are the basis for effective computational algorithms for validating existence and uniqueness of solutions
- Know what is interval and what is point
- Use Mean Value, centered form, or slope representations
- If you can readily compute two enclosures, intersect
- Approximate, then validate

Interval Grand Challenges - Neumaier

1. Large-scale global optimization. Although this is NP-hard, so is mixed integer programming; nevertheless large mixed integer programs are solved routinely (though not all of them). Global optimization is the only area of wide practical applicability where intervals are likely to have a major impact in the near future.
2. Error bounds for realistic discrete finite element problems with realistic (dependent) uncertainties. Applications abound, and interval methods could become competitive, as replacement for Monte-Carlo studies.
3. Solution of large and sparse linear equations with interval coefficients. Apart from a paper by Rump, nothing has been done, although sparse matrices are ubiquitous in applications.
4. Error bounds for large systems of ODE's. Both AWA and COSY which were recently under discussion, only handle very small systems.

5. Error bounds for stiff systems and singularly perturbed systems of ODE's. The books by Hairer and Wanner contain enough analytic results that could be used to get a start, by creating variants of the results with constructively verifiable assumptions.
6. Error bounds for elliptic partial differential equations on irregular meshes. Most applications have their mesh adapted to the problem; current interval work is mostly on toy problems with very simple domains.

Wrap-up

How did we do?

At the conclusion of this tutorial, you should be able to

- Understand most of the how and why of interval algorithms
- Define your own f to use in existing software
- Read modern papers on validating algorithms
- Adapt and extend them for your own use
- Write your own algorithms and papers

Appendix A: ex6_bp.v.f95

$$X_n = \int_0^1 x^n \exp(x-1) dx$$

$$X_n = 1 - nX_{n-1}$$

Reference: I. Babuska, M. Prager, and E. Vitasek, Numerical Processes in Differential Equations, Interscience, 1966.

Illustrates: Accumulation of rounding error

```
PROGRAM EX6_BP
  INTEGER  :: N = 1
  REAL     :: X
  INTERVAL :: IVL_X
  CHARACTER*30 BUFFER
  PRINT *, "X_N = integral_0^1 x**N exp(x-1) dx"
  PRINT *, "      = 1 - N * X_{N-1}, (Babuska, Prager, & Vitasek)"
  PRINT *, "Enter an interval, e.g., [0.3678, 0.3679], 0.3678794411"
  WRITE(*, 1, ADVANCE='NO')
  READ(*, '(A12)', IOSTAT=IOS) BUFFER
  READ(BUFFER, '(Y12.16)') IVL_X
  X = MID(IVL_X)
  PRINT *, "          N          X          Interval X"
  DO WHILE (WID(IVL_X) <= 100)
    PRINT '(I10, F20.12, 2X, Y25.12)', N, X, IVL_X
    N = N + 1
    X = 1 - N * X
    IVL_X = 1 - N * IVL_X
  END DO
  1 FORMAT(" X = ? ")
END PROGRAM EX6_BP
```

Result:

```
ex6
X_N = integral_0^1 x**N exp(x-1) dx
      = 1 - N * X_{N-1}, (Babuska, Prager, & Vitasek)
Enter an interval, e.g., [0.3678, 0.3679], 0.3678794411
X = ? 0.3678794411
      N          X          Interval X
      1          0.367879450321      0.367879441
      2          0.264241099358      0.264241118
      3          0.207276701927      0.20727664
      4          0.170893192291      0.17089341
      5          0.145534038544      0.1455329
      6          0.126795768738      0.1268024
      7          0.112429618835      0.112383
      8          0.100563049316      0.10093
      9          0.094932556152      0.0916
     10          0.050674438477      0.084
     11          0.442581176758      0.07
     12          -4.310974121094      [.58094E-001, .15390 ]
     13          57.042663574219      [-1.001 , 0.2448 ]
     14          -797.597290039062      [-2.427 , 15.01 ]
```

Appendix B: ex6_bp.v.m

$$X_n = \int_0^1 x^n \exp(x-1) dx$$

$$X_n = 1 - nX_{n-1}$$

Reference: I. Babuska, M. Prager, and E. Vitasek, Numerical Processes in Differential Equations, Interscience, 1966.

Illustrates: Accumulation of rounding error

```
intvalinit('Display_',0)
n = 1
x = 0.3678794411
ivl_x = intval('0.3678794411_')

while n < 16
  n = n + 1
  x = 1 - n * x
  ivl_x = 1 - n * ivl_x
end;
```

Result (lightly edited):

```
>> ex6_bpv
Interval X"
n = 1 x = 0.36787944110000 ivl_x = 0.3678794411_____
n = 2 x = 0.26424111780000 ivl_x = 0.264241118_____
n = 3 x = 0.20727664660000 ivl_x = 0.20727665_____
n = 4 x = 0.17089341360000 ivl_x = 0.17089341_____
n = 5 x = 0.14553293200000 ivl_x = 0.1455329_____
n = 6 x = 0.12680240800001 ivl_x = 0.1268024_____
n = 7 x = 0.11238314399991 ivl_x = 0.112383_____
n = 8 x = 0.10093484800073 ivl_x = 0.10093_____
n = 9 x = 0.09158636799346 ivl_x = 0.0916_____
n = 10 x = 0.08413632006535 ivl_x = 0.084_____
n = 11 x = 0.07450047928110 ivl_x = 0.07_____
n = 12 x = 0.10599424862681 ivl_x = 0.1_____
n = 13 x = -0.37792523214858 ivl_x = -0.1_____
```

```

n = 14  x = 6.29095325008018  ivl_x = 0_----- L = A(J,I)/S
n = 15  x = -93.36429875120268  ivl_x = 1.0e+002 * -0_----- DO K=I1,Dim
                                     A(J,K) = A(J,K)-L*A(I,K)
                                     END DO
                                     B(J) = B(J)-L*B(I)
                                     END DO
END DO

```

Appendix C. Sample interval Gaussian elimination

From Sun's Fortran 95
/opt/SUNWsprow/examples/intervalmath/general/IA_Gauss
See that directory for the complete context

```

SUBROUTINE I_Gauss(Dim,A,B,X,Sgn)
!
!      Running interval Gauss method
!      with the mignitude pivoting
!
INTERVAL(8) L, R, S, A(50,50), B(50), X(50)
INTEGER Dim
LOGICAL Sgn

```

```

DO I=1,Dim
  I1 = I+1
  ! detecting the mignitude maximum
  S = A(I,I)
  SM = MIG(S)
  K = I
  DO J=I1,Dim
    R = A(J,I)
    RM = MIG(R)
    IF( RM<=SM ) CYCLE
    S = R
    SM = RM
    K = J
  END DO

```

Appendix C. Subroutine I_Gauss (continued)

```

IF( SM==0. ) THEN
  Sgn = .FALSE.
  RETURN
END IF

IF( K/=I ) THEN
  ! rearranging the rows if necessary
  DO J=I,Dim
    R = A(I,J)
    A(I,J) = A(K,J)
    A(K,J) = R
  END DO
  R = B(I)
  B(I) = B(K)
  B(K) = R
END IF

! recalculating the rest of the matrix entries
DO J=I1,Dim

```

```

!      performing the backward substitution
DO I=Dim,1,-1
  S = B(I)
  DO J=I+1,Dim
    S = S-A(I,J)*X(J)
  END DO
  X(I) = S/A(I,I)
END DO

END SUBROUTINE I_Gauss

```

Appendix D. Hybrid dense linear system solver

From TU Hamburg-Harburg INTLAB for MATLAB
.../toolbox/intlab/intval/verifylss.m

```

function X = verifylss(A,b)
%VERIFYLSS   Verified solution of linear system
%
%   X = verifylss(A,b)
%
%Hybrid dense linear system solver. First stage based on Krawc
% R. Krawczyk: Newton-Algorithmen zur Bestimmung von Nullstel
% Fehlerschranken, Computing 4, 187-201, 1969.
% R.E. Moore: A Test for Existence of Solutions for Non-Linear
% SIAM J. Numer. Anal. 4, 611-615, 1977.
%with modifications for enclosing the error with respect to an
%solution, an iteration scheme and epsilon-inflation.

```

```

function X = denselss(A,b) % linear system solver for d

n = dim(A);
midA = mid(A);
midb = mid(b);

% preconditioner: approximate inverse
R = inv( midA ) ;

% approximate solution with one iteration to ensure backward
xs = R * midb ;
xs = xs + R*(midb - midA*xs);

% interval iteration
A = intval(A);
Z = R * ( b - A*xs ) ;
RA = R*A;
C = speye(n) - RA;
Y = Z;

```

```
E = 0.1*rad(Y)*hull(-1,1) + midrad(0,10*realmin);
k = 0; kmax = 7; ready = 0;
```

Appendix D. verifylss (continued)

```
while ( ~ready ) & ( k<kmax ) & ( ~any(isnan(Y(:))) )
    k = k+1;
    X = Y + E;
    Y = Z + C * X;
    ready = all(all(in0(Y,X)));
end

if ready % success first stage
    X = xs + Y;
    return
end

% second stage starts
b = R*b;
dRA = diag(RA);
A = compmat(RA);
B = inv(A);

v = abs(B*ones(n,1));

setround(-1)
u = A*v;
if all(min(u)>0),
    dAc = diag(A);
    A = A*B - speye(n);
    setround(1)
    w = max(-A./(u*ones(1,n))); % w_k = max_i{ -A(i,:) / (u(i)) }
    dlow = v.*w' - diag(B);
    dlow = max(0,-dlow);
    B = B + v*w;
    u = B*abs(b);
    d = diag(B);
    alpha = dAc + (-1)./d;
    k = size(b,2);
    if k==1
        beta = u./dlow - abs(b);
        X = ( b + midrad(0,beta) ) ./ ( dRA + midrad(0,alpha) );
    else
        % d and dRA adapted for multiple r.h.s.
        v = ones(1,k);
        beta = u./(d*v) - abs(b);
        X = ( b + midrad(0,beta) ) ./ ( ( dRA + midrad(0,alpha) ) );
    end
    return
end;

% second stage failed, compmat(RA) is numerically not an H-matrix
X = repmat(NaN,size(b));
```

Verified solution of nonlinear system
From TU Hamburg-Harburg INTLAB for MATLAB
.../toolbox/intlab/intval/verifynlss.m

```
function [ X , xs , k ] = verifynlss(f,xs,slopes,see,varargin)
%VERIFYNLSS Verified solution of nonlinear system
%
% [ X , xs , k ] = verifynlss(f,xs,slopes,see,P1,P2,...)
%
% f is name of function, to be called by f(xs), xs is approximation
%
% optional input slopes 'g' use gradient, proves uniqueness
% 's' use slopes, better, but w/o uniqueness
% see see intermediate results
% P1,... extra parameters for function evaluation
% f(x,P1,P2,...)
%
% optional output xs improved approximation (column vector)
% k interval iteration steps
%
%
% Simple, one-dimensional nonlinear functions which can be written as
% formula string, can be entered directly. The unknown must be x.
% X = verifynlss('x*exp(x)-1',.6)
% evaluates an inclusion of the zero of x*exp(x)-1=0 near xs=.6
%
% Nonlinear system solver based on the Krawczyk operator, see
% R. Krawczyk: Newton-Algorithmen zur Bestimmung von Nullstellen mit
% Fehlerschranken, Computing 4, 187-201, 1969.
% R.E. Moore: A Test for Existence of Solutions for Non-Linear Equations,
% SIAM J. Numer. Anal. 4, 611-615, 1977.
% with modifications for enclosing the error with respect to an interval
% solution, an iteration scheme, epsilon-inflation and an improved
% iteration.
% Using slopes verifies existence and uniqueness of a zero of a
% function within the inclusion interval. This also implies multiple
% the zero, and nonsingularity of the Jacobian at the zero.
% Using slopes implies existence but not uniqueness of a zero.
% inclusion of zero clusters and multiple zeros. For details, see
% S.M. Rump: Inclusion of Zeros of Nowhere Differentiable n-Dimensional
% Functions, Reliable Computing, 3:5-16 (1997).
%
% written 10/16/98 S.M. Rump
% modified 10/12/99 S.M. Rump output NaN-vector in case of failure
% interval iteration stops if no zero is found
%
% xs = xs(:);
% if ( nargin<3 ) | isempty(slopes)
%     slopes = 0;
% else
%     if ~ischar(slopes)
%         error('third parameter of verifynlss must be char')
%     end
%     slopes = isequal(lower(slopes(1)), 's');
% end
%
% if ( nargin<4 ) | isempty(see)
%     see = 0;
% end
```

Appendix E. verifynlss.m

```

% Convert to inline function as needed
strfun = fcnchk(f,length(varargin));

% floating point Newton iteration
n = length(xs);
xsold = xs;
k = 0;
while ( norm(xs-xsold)>1e-10*norm(xs) & k<10 ) | k<1
    k = k+1; % at most 10, at least 1 iteration performed
    xsold = xs;
    y = feval(strfun,gradientinit(xs),varargin{:});
    if see
        disp(['residual norm(f(xs_k)), floating point iteration ',
            sprintf('%d',k),
            norm(y.x)
        end
        xs = xs - y.dx\y.x;
    end

% interval iteration
R = inv(y.dx);
Z = - R * feval(strfun,intval(xs),varargin{:});
X = Z;
E = 0.1*rad(X)*hull(-1,1) + midrad(0,realmin);
ready = 0; k = 0; kmax = 10;
while ( ~ready ) & ( k<kmax ) & ( ~any(isnan(X)) )
    k = k+1;
    if see
        disp(['interval iteration ' sprintf('%d',k)])
    end
    Y = hull( X + E , 0 ); % epsilon inflation
    Yold = Y;
    if slopes % use slopes
        x = slopeinit(xs,xs+Y);
        y = feval(strfun,x,varargin{:});
        C = eye(n) - R * y.s; % automatic slopes
    else % use gradients
        x = gradientinit(xs+Y);
        y = feval(strfun,x,varargin{:});
        C = eye(n) - R * y.dx; % automatic gradients
    end
    i=0;
    while ( ~ready ) & ( i<2 ) % improved interval iteration
        i = i+1;
        X = Z + C * Y;
        ready = all(all(in0(X,Y)));
        Y = intersect(X,Yold);
    end
end
if ready
    X = xs+Y; % verified inclusion
else
    X = repmat(NaN,n,1); % inclusion failed
end

```

Appendix F. Interval_Newton.m

One-dimensional interval Newton for multiple roots

```
function [ root, countRoots ] = Interval_Newton (f, Tolerance,
```

```

topOfPile, Pile)

strfun = fcnchk(f);
countRoots = 0;
root(1) = intval(0);
while topOfPile > 0
    topOfPile
    pendingInterval = Pile(topOfPile)
    fromINewton = OneINewton (strfun, pendingInterval)
    if isempty(fromINewton)
        disp('EMPTY');
    else
        if length(fromINewton) == 1
            if diam(fromINewton) < Tolerance
                disp('FOUND root:'); fromINewton
                countRoots = countRoots + 1;
                root(countRoots) = fromINewton;
            else % Not yet converges/rejected
                topOfPile = topOfPile + 1;
                Pile(topOfPile) = fromINewton;
            end;
        else % TWO subintervals
            topOfPile = topOfPile + 1;
            Pile(topOfPile) = fromINewton(1);
            topOfPile = topOfPile + 1;
            Pile(topOfPile) = fromINewton(2);
        end;
    end;
end;

% File: test_Interval_Newton.m
%
% Author: George Corliss, Marquette University, May 2002.

q = '-400*(1.7 - x^2)*x + 2*x + 17'
Tolerance = 1.0e-10

disp(' '); disp('No root in');
X = intval('[2.1, 2.2]')
topOfPile = 1; Pile(topOfPile) = X;
[ root, countRoots ] = Interval_Newton (q, Tolerance, topOfPile);
reportPile ( countRoots, root );
disp('No root');
disp(' '); disp('Press [Enter] for next test');
pause;

disp('One root in');
X = intval('[1, 2]')
topOfPile = 1; Pile(topOfPile) = X;
[ root, countRoots ] = Interval_Newton (q, Tolerance, topOfPile);
reportPile ( countRoots, root );
disp('Note: Converged');
disp(' '); disp('Press [Enter] for next test');
pause;

disp('THREE roots in');
X = intval('[-1.5, 1.5]')
topOfPile = 1; Pile(topOfPile) = X;

```

```
[ root, countRoots ] = Interval_Newton (q, Tolerance, topOfPile, PendingInterval, Newton
reportPile ( countRoots, root );
disp('Note: Have resolved three roots');
```

```
q = -400*(1.7 - x^2)*x + 2*x + 17
Tolerance = 1.0000000000000000e-010

No root in
intval X = [ 2.099999999999998, 2.200000000000001]
topOfPile = 1
intval pendingInterval = [ 2.099999999999998, 2.200000000000000]
intval fromINewton = [ NaN,
EMPTY
```

Appendix F. OneINewton.m

```
function X = OneINewton (strfun, trial_inclusion)
X = trial_inclusion;
pt_x = mid(X);
pt_y = feval(strfun, pt_x);
Y = feval(strfun, gradientinit(X));
del_X = Extended_divide (intval(pt_y), Y.dx);
new_X = Extended_subtract (pt_x, del_X);
X = Extended_intersect (X, new_X);
```

```
Report of 0 roots found:
EMPTY - No roots
```

No root

Press [Enter] for next test

```
One root in
intval X = [ 1.000000000000000, 2.000000000000000]
topOfPile = 1
intval pendingInterval = [ 1.000000000000000, 2.000000000000000]
intval fromINewton = [ 1.000000000000000, 1.4150897622]
topOfPile = 1
intval pendingInterval = [ 1.000000000000000, 1.4150897622]
intval fromINewton = [ 1.26400699773200, 1.3941267406]
topOfPile = 1
intval pendingInterval = [ 1.26400699773200, 1.3941267406]
intval fromINewton = [ 1.28471114692707, 1.2958395988]
topOfPile = 1
intval pendingInterval = [ 1.28471114692707, 1.2958395988]
intval fromINewton = [ 1.28918606321756, 1.2892141392]
topOfPile = 1
intval pendingInterval = [ 1.28918606321756, 1.2892141392]
intval fromINewton = [ 1.28919889507851, 1.2891988951]
FOUND root:
intval fromINewton = [ 1.28919889507851, 1.28919889515811]
```

Report of 1 roots found:

```
intval ans = [ 1.28919889507851, 1.28919889515811]
```

Note: Converged

Press [Enter] for next test

```
THREE roots in
intval X = [ -1.500000000000001, 1.500000000000001]
topOfPile = 1
intval pendingInterval = [ -1.500000000000001, 1.500000000000000]
intval fromINewton = [ -1.500000000000001, -0.0084075173]
[ 0.00686037126715, 1.500000000000000]
topOfPile = 2
intval pendingInterval = [ 0.00686037126715, 1.500000000000000]
intval fromINewton = [ 0.91304921869821, 1.500000000000000]
[ 0.00686037126715, 0.2773585451]
topOfPile = 3
intval pendingInterval = [ 0.00686037126715, 0.2773585451]
intval fromINewton = [ 0.00858713493036, 0.0267573023]
topOfPile = 3
intval pendingInterval = [ 0.00858713493036, 0.0267573023]
intval fromINewton = [ 0.02507796899022, 0.0250863975]
topOfPile = 3
intval pendingInterval = [ 0.02507796899022, 0.0250863975]
```

```
function z = Extended_divide (a, b)
if in(0, b)
if 0 == inf(b)
z = a * hull (intval(1)/sup(b), inf);
elseif 0 == sup(b)
z = a * hull (-inf, intval(1)/inf(b));
elseif in (0, a)
z = [-inf, inf];
else
z = [ (a * hull (intval(1)/sup(b), inf)),
(a * hull (-inf, intval(1)/inf(b))) ];
end;
else
z = a / b;
end;
```

```
function z = Extended_subtract (a, b)
if length(b) == 1
z = a - b;
else
z = [ a - b(1), a - b(2) ];
end;
```

```
function z = Extended_intersect (a, b)
if length(b) == 1
z = intersect(a, b);
else
a1 = intersect(a, b(1));
a2 = intersect(a, b(2));
if isempty(a1)
z = a2;
elseif isempty(a2)
z = a1;
else
z = [a1, a2];
end;
end;
```

Appendix F. Interval_Newton.m results (lightly edited)

intval fromINewton = [0.02508305678396, 0.02508305678462]

FOUND root:

intval fromINewton = [0.02508305678396, 0.02508305678462]

topOfPile = 2

intval pendingInterval = [0.91304921869821, 1.5000000000000001]

intval fromINewton = [1.25523269417695, 1.5000000000000001]

topOfPile = 2

intval pendingInterval = [1.25523269417695, 1.5000000000000001]

intval fromINewton = [1.27143697805951, 1.31393335698275]

topOfPile = 2

intval pendingInterval = [1.27143697805951, 1.31393335698275]

intval fromINewton = [1.28903319230460, 1.28937866432852]

topOfPile = 2

intval pendingInterval = [1.28903319230460, 1.28937866432852]

intval fromINewton = [1.28919889231842, 1.28919889802910]

topOfPile = 2

intval pendingInterval = [1.28919889231842, 1.28919889802910]

intval fromINewton = [1.28919889511659, 1.28919889511660]

FOUND root:

intval fromINewton = [1.28919889511659, 1.28919889511660]

topOfPile = 1

intval pendingInterval = [-1.5000000000000001, -0.00840751730959]

intval fromINewton = [-1.5000000000000001, -0.93063633833926]

intval fromINewton = [-0.22796288138712, -0.00840751730959]

topOfPile = 2

intval pendingInterval = [-0.22796288138712, -0.00840751730959]

intval fromINewton = [NaN, NaN]

EMPTY

topOfPile = 1

intval pendingInterval = [-1.5000000000000001, -0.93063633833926]

intval fromINewton = [-1.5000000000000001, -1.27613745412581]

topOfPile = 1

intval pendingInterval = [-1.5000000000000001, -1.27613745412581]

intval fromINewton = [-1.33284344191375, -1.30057247843348]

topOfPile = 1

intval pendingInterval = [-1.33284344191375, -1.30057247843348]

intval fromINewton = [-1.31437395648014, -1.31419787808937]

topOfPile = 1

intval pendingInterval = [-1.31437395648014, -1.31419787808937]

intval fromINewton = [-1.31428195270806, -1.31428195112908]

topOfPile = 1

intval pendingInterval = [-1.31428195270806, -1.31428195112908]

intval fromINewton = [-1.31428195190093, -1.31428195190092]

FOUND root:

intval fromINewton = [-1.31428195190093, -1.31428195190092]

Report of 3 roots found:

intval ans = [0.02508305678396, 0.02508305678462]

intval ans = [1.28919889511659, 1.28919889511660]

intval ans = [-1.31428195190093, -1.31428195190092]

Note: Have resolved three roots

ODE Problem

AWA Algorithm:

- loop for each integration step
- Algorithm I: Validate existence and uniqueness
- Algorithm II: Compute tighter enclosure

Ordinary differential equations

The purpose of this section is to provide an introduction to interval techniques for the validated computation of solutions to initial value problems in ordinary differential equations (ODE's) $u' = f(t, u)$, $u(t_0) = u_0$. We show from the Picard Iteration Theorem that validated enclosures are Computing guaranteed bounds for the range of f over a region containing (t_0, u_0) leads to an interval-valued function $[u](t)$ that encloses the true solution $u(t)$ in a neighborhood of t_0 .

There are several excellent surveys of interval techniques for ODE's including [Bauch 1989], [Corliss 1989], [Corliss 1994], [Nickel 1986], [Rihm 1994], [Stetter 1986], and [Stetter 1990]. Extensive bibliographies to literature in the field may be found in the survey papers and in [Corliss 1988] or [Corliss 1991].

The most widely used software for the validated solution of ODE's is AWA (Anfangswertaufgabe) by Rudolf Lohner (1978, 1987, and following). Lohner attempts to minimize the wrapping effects by representing the enclosure of the solution at each time step as a linear transformation of an interval: $u(t_j) \in A_0 \cdots A_{j-1}[x_j]$ at time $t = t_j$. He advances the solution to $t = t_{j+1} := t_j + h$ using two algorithms:

Algorithm I. Validate existence and uniqueness using Picard-Bindelöf iteration

Input: Problem: $u' = f(t, u)$ Final time: t_f
Initial conditions: $t_j, u(t_j) \in \mathbf{u}_j$ Tolerance

Output: Coarse enclosure: \mathbf{u}_{j_0} Step size: h

Output assertions:
 $u' = f(t, u)$, $u(t_j) = u_j \in \mathbf{u}_j$ has a unique solution in $[t_j, t_j + h] \times \mathbf{u}_{j_0}$
 $u(t) \in \mathbf{u}_{j_0}$ for all $t \in [t_j, t_j + h]$

Algorithm II. Compute tighter enclosure using Taylor series and local coordinate transformation

Input: Results from Algorithm I

Output: Matrix: A_j

Appendix G: Ordinary differential equations (ODE's)

Communicate a feeling for the strategies of computing validated bounds on solutions to initial value problems in

Interval: \mathbf{x}_{j+1}
 Next node: $t_{j+1} \leq t_j + h$
 Solution: $\mathbf{u}_{j+1} :=$
 Convex hull ($A_0 A_1 \cdots A_j \mathbf{x}_{j+1}$)
 with $w(\mathbf{u}_{j+1}) \leq \text{Tolerance}$
 Output assertion:
 $u(t_{j+1}) \in A_0 A_1 \cdots A_j \mathbf{x}_{j+1}$

We give a worked-out numerical example.

ODE: Problem

Numerical solution for ODE's

$$u' = f(t, u), \quad u(t_0) = u_0 \in [u_0]$$

Example – Lorenz system

$$\begin{aligned} x' &= \sigma(y - x) & x(0) &= 10 \\ y' &= rx - y - xz & y(0) &= 10 \\ z' &= xy - bz & z(0) &= 10 \\ \sigma &= 10, b = 8/3, r = 28 \end{aligned}$$

What is a “Validated Solution” of an ODE?

- Guarantee that there exists a unique solution u in the interval $[t_0, t_f]$
- Compute an interval-valued function $\mathbf{u}(t) = \hat{\mathbf{u}}(t) + \mathbf{e}(t)$ such that

$$\begin{aligned} u(t, u_0) &\in \mathbf{u}(t), \text{ and} \\ w(\mathbf{u}(t)) &\leq \text{Tolerance}, t \in [t_0, t_f], u_0 \in \mathbf{u}_0, \end{aligned}$$

or equivalently,

- Compute an approximate solution $\hat{\mathbf{u}}(t)$ and an interval-valued error function $\mathbf{e}(t)$ such that

$$\begin{aligned} u(t, u_0) &\in \hat{\mathbf{u}}(t) + \mathbf{e}(t), \text{ and} \\ w(\mathbf{e}(t)) &\leq \text{Tolerance}, t \in [t_0, t_f], u_0 \in \mathbf{u}_0 \end{aligned}$$

ODE: Existence and Uniqueness

Classical theory of existence and uniqueness of solutions for general, nonlinear ODE's can be found in most standard texts (see eg. [Ince 1926] or [Butcher 1987])

Let f be defined and continuous in a rectangular domain $D := \{(t, u) \in \mathbb{R} \times \mathbb{R}^n : |t - t_0| \leq a, \|u - u_0\| \leq b\}$. Then f satisfies a *Lipschitz condition* in u if there exists a Lipschitz constant L such that

$$\|f(t, u) - f(t, v)\| \leq L\|u - v\|, \text{ and } (t, u), (t, v) \in D$$

Existence and Uniqueness Theorem. Let f be continuous and Lipschitz in D , and let $M := \max_{(t,u) \in D} \|f(t, u)\|$.

Then there exists a unique solution $u(t)$ for $|t - t_0| \leq \min(a, M/b)$.

Suffices to verify $[u_0]_1 \subseteq [u_0]_0$:

Theorem [Walter1970a, Lohner1988a]. If we can find intervals $[t_0, t_0 + h]$ and $[u_0]_0$ such that

$$[u_0]_1 := u_0 + [0, h] * f([t_0, t_0 + h], [u_0]_0) \subseteq [u_0]_0,$$

then the ODE has a unique solution.

ODE: Existence and Uniqueness

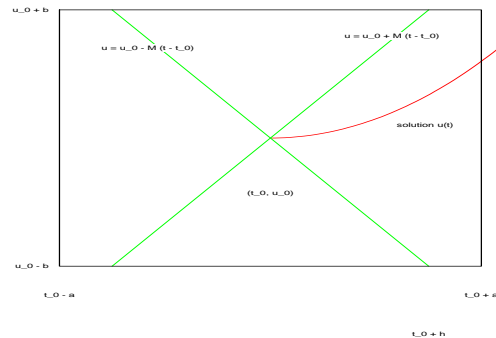
If our computer programs can

1. Find an interval $[t_0, t_1]$ and an interval $[u_0] = [u_0 - b, u_0 + b] \supseteq u_0$ for which
2. f is continuous in $D := [t_0, t_1] \times [u_0]$,
3. $\|f\| \leq M \in D$, and
4. $t_1 = \min(t_1, t_0 + b/M)$,

then we can guarantee that the ODE has a unique solution.

ODE: Fig. 1. Existence and Uniqueness

Existence and Uniqueness Theorem



ODE: Algorithm I. Validate existence and uniqueness

Input:

Problem: $u' = f(t, u)$
 Initial conditions: $t_j, u(t_j) \in \mathbf{u}_j$
 Final time: t_f
 Tolerance: Tolerance

Input assertions:

Width $w(\mathbf{u}_j) < \text{Tolerance}$
 $f(t_j, \mathbf{u}_j)$ can be evaluated with no $\sqrt{0}$ or fractional powers of non-positive numbers.

Output:

Either

Coarse enclosure: \mathbf{u}_{j_0}

Step size: h

or else

Failure

Output assertions:

$u' = f(t, u)$, $u(t_j) = u_j \in \mathbf{u}_j$ has a unique solution

in $[t_j, t_j + h] \times \mathbf{u}_{j_0}$

$u(t) \in \mathbf{u}_{j_0}$ for all $t \in [t_j, t_j + h]$

Basic idea:

Picard-Lindelöf iteration

ODE: Algorithm I. Validate existence and uniqueness

Apply Banach Fixed Point Theorem to Picard-Lindelöf operator mapping $T_j := [t_j, t_j + h] \rightarrow T_j$

$$\Phi(u)(t) := u_j + \int_{t_j}^t f(\tau, u(\tau)) d\tau$$

Algorithm I:

1. Guess an *a priori* enclosure \mathbf{u}_{j_0} ;
Guess a step size h ;
2. $\mathbf{u}_{j_1} := \Phi(\mathbf{u}_{j_0})$;
3. If $\mathbf{u}_{j_1} \subseteq \mathbf{u}_{j_0}$
then Algorithm I is successful
4. else try again

ODE: Step I.1. Guess *a priori* enclosure \mathbf{u}_{j_0} and a step size h .

AWA uses $[u_j]_0 := [u_j] + [0, h][f(t_j, [u_j])] + h\beta[-1, 1]$

Bound f at the initial condition: $[f(t_j, [u_j])]$

Inflate by β

Extend over the interval $T_j := [t_j, t_j + h]$

Compute bounds

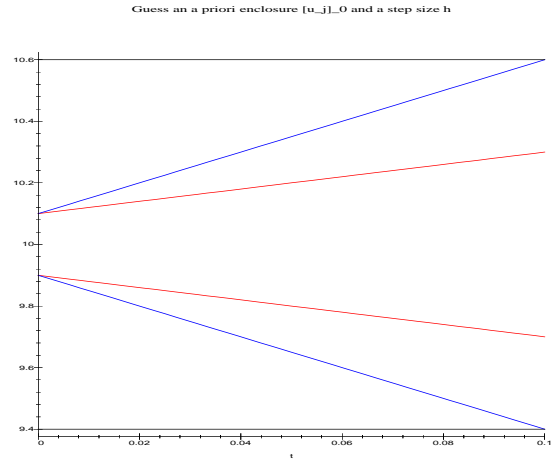
For Lorenz system at the conclusion of Step 1,

$$[u_j]_0 = \begin{pmatrix} [9.399999975, 10.60000005] \\ [9.599999997, 27.88900010] \\ [9.599999997, 17.96100006] \end{pmatrix}$$

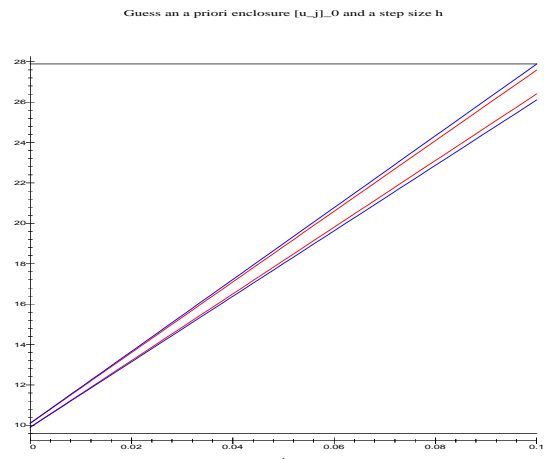
$$h = 0.1$$

ODE: Fig. 2. Guess *a priori* enclosure and step size # 3

ODE: Fig. 4. Guess *a priori* enclosure and step size # 1



ODE: Fig. 5. Guess *a priori* enclosure and step size # 2



ODE: Step I.2. $[u_j]_1 := \Phi([u_j]_0)$

Various ways we might compute an enclosure for

$$\Phi(u)(t) := u_j + \int_{t_j}^t f(\tau, u(\tau)) d\tau$$

Linear *a priori* enclosure:

$$[u_j]_1(t) := [u_j] + [f(T_j, [u_j]_0)](t - t_j)$$

Bound the range of f over the domain D (in Maple):

RangeOf_f := map (i -> inapply (i, x, y, z)(U_j_0), f)

$$\text{RangeOf}_f = \begin{pmatrix} [-10.00000065, 184.8900015] \\ [44.92439729, 196.9600019] \\ [42.34399953, 270.0234027] \end{pmatrix}$$

```

for i from 1 to NumberOfEquations do
  U_j_1[i] := [ U_j[i][Inf] + RangeOf_f[i][Inf] * (t - t_j),
               U_j[i][Sup] + RangeOf_f[i][Sup] * (t - t_j) ];
od;

```

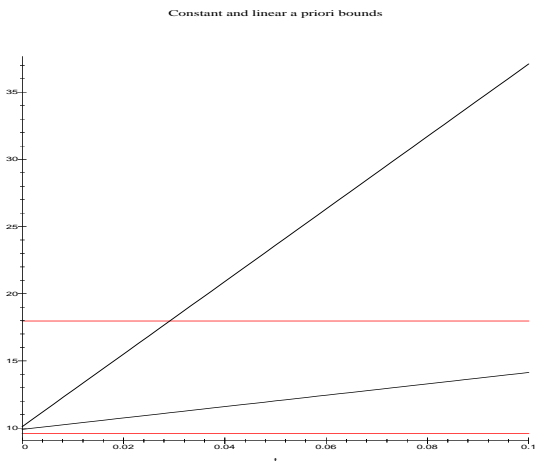
$$[u_j]_1 = \begin{pmatrix} [9.899999999 - 10.00000065 t, \\ 10.10000001 + 184.8900015 t] \\ [9.899999999 + 44.92439729 t, \\ 10.10000001 + 196.9600019 t] \\ [9.899999999 + 42.34399953 t, \\ 10.10000001 + 270.0234027 t] \end{pmatrix}$$

ODE: Step I.3. If $u_{j_1} \subseteq u_{j_0}$ then Algorithm I is successful

When do the linear bounds exit the region?
 Component 1 at about $t = 0.003$
 $[u_j]_0[1] = [9.399999975, 10.60000005]$
 $[u_j]_1[1] =$
 $[9.899999999 - 10.00000065 t, 10.10000001 + 184.8900015 t]$
 solve (U_j_1[1][Sup] = U_j_0[1][Sup]);

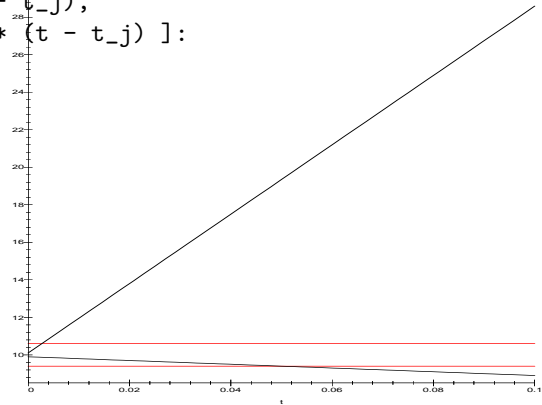
0.002704310866

ODE: Fig 6. Constant and linear *a priori* bounds # 3



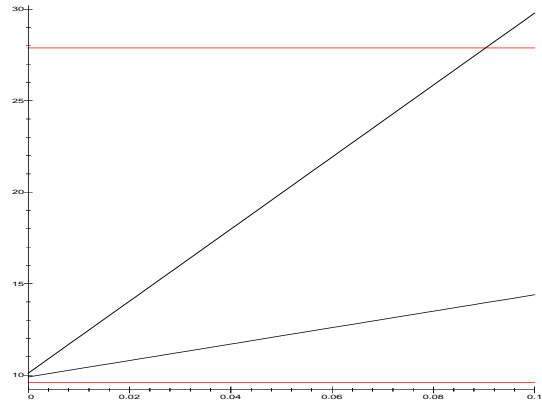
ODE: Fig 7. Constant and linear *a priori* bounds # 1

Constant and linear a priori bounds



ODE: Fig 8. Constant and linear *a priori* bounds # 2

Constant and linear a priori bounds



ODE: Step I.3. If $u_{j_1} \subseteq u_{j_0}$ then Algorithm I is successful

We have validated that for

$$t \in T_j := [t_j, t_j + h] = [0, 0.002704310866],$$

the components of the solution for the Lorenz system

1. exist,
2. are unique, and
3. lie in the constant bound enclosure

$$[u_j]_0 = \begin{pmatrix} [9.399999975, 10.60000005] \\ [9.599999997, 27.88900010] \\ [9.599999997, 17.96100006] \end{pmatrix}$$

and in the linear enclosure

$$[u_j]_1 = \begin{pmatrix} [9.899999999 - 10.00000065 t, \\ 10.10000001 + 184.8900015 t] \\ [9.899999999 + 44.92439729 t, \\ 10.10000001 + 196.9600019 t] \\ [9.899999999 + 42.34399953 t, \\ 10.10000001 + 270.0234027 t] \end{pmatrix}$$

ODE: Step I.3. If $u_{j_1} \subseteq u_{j_0}$ then Algorithm I is successful

At this point, we have several alternatives:

1. Accept the *a priori* bound

$$[u_j]_0 = \begin{pmatrix} [9.399999975, 10.60000005] \\ [9.599999997, 27.88900010] \\ [9.599999997, 17.96100006] \end{pmatrix}$$

on the interval

$$T_j := [0, 0.002704310866]$$

and go on to AWA Algorithm II. That is what we shall do here.

2. Try again with a wider guess for the *a priori* bound and try to validate enclosure over a longer interval. That is what the current version of AWA does.
3. Try again with a narrower guess for the *a priori* bound and try to validate enclosure over a longer interval.
4. Try a more sophisticated method for bounding $\Phi(u)$. Lohner currently does this by computing a priori bounds for $u^{(p)}(t)$.

ODE: Algorithm II. Compute tighter enclosure

Input:

Problem: $u' = f(t, u)$
 Initial conditions: $t_j, u(t_j) \in u_j$
 Current solution: matrices A_0, A_1, \dots, A_{j-1}
 and interval x_j
 Coarse enclosure: u_{j_0}
 Step size: h
 Tolerance: Tolerance

Input assertion:

$$u(t) \in u_{j_0} \text{ for all } t \in [t_j, t_j + h]$$

Output:

Either

Matrix: A_j

Interval: x_{j+1}
 Next node: $t_{j+1} \leq t_j + h$
 Solution: $u_{j+1} := \text{Convex hull}(A_0 A_1 \dots A_j x_{j+1})$

or else

Failure

Also available:

Approximation solution: $\hat{u}(t)$

Continuous enclosure on each subinterval: $u_{j_1}(t)$

Output assertion:

$$u(t_{j+1}) \in A_0 A_1 \dots A_j x_{j+1}$$

$$w(u_{j+1}) \leq \text{Tolerance}$$

Basic idea:

Truncated Taylor series plus the remainder term

Local coordinate transformations

ODE: Naive Taylor Enclosure

Local Taylor expansion encloses u :

$$\begin{aligned} u(t) &\in u_{j_1}(t) \\ &:= u_j + u'(t_j)(t - t_j) + \dots \\ &\quad + \frac{u^{(p-1)}(t_j)}{(p-1)!}(t - t_j)^{p-1} + \frac{u^{(p)}(T_j)}{p!}(t - t_j)^p \end{aligned}$$

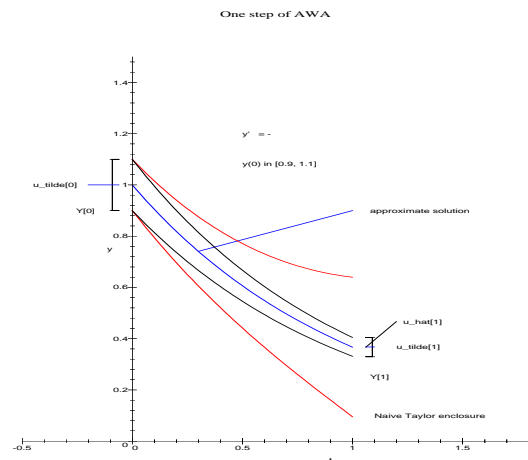
Derivatives are computed by automatic differentiation

Let $u_{j+1} := u_{j_1}(t_j + h)$

Gives width $(u_{j+1}) > \text{width}(u_j)$

Example: $y' = -y, y_0 \in [0.9, 1.1]$

ODE: Fig F. Naive Taylor enclosure does not contract



ODE: Algorithm II. Compute tighter enclosure

Point: Enclosures computed by naive Taylor enclosure do not contract, even when the true trajectories are strongly contracting

Need to be more clever.

[Eijgenraam 1981] and [Lohner 1978, 1987] express the enclosure as

$$u(t) \in \left(\begin{array}{l} \text{point-valued truncated Taylor} \\ \text{series for approximate solution} \end{array} \right) + \left(\begin{array}{l} \text{old} \\ \text{error} \end{array} \right) + \left(\begin{array}{l} \text{enclosure of local} \\ \text{truncation error} \end{array} \right)$$

AWA follows contractive flows and controls the wrapping effect

Algorithm II:

1. Point-valued approximate solution $\hat{u}_j(t)$;
2. Enclosure \mathbf{z}_{j+1} of the local error of $\hat{u}_j(t)$;
3. Enclosure for the transformation matrix \mathbf{A}_j ;
4. Prepare for next solution step;

ODE: Step II.1. Point-valued approximate solution $\hat{u}_j(t)$

Choose a point $\tilde{u}_j \in \mathbf{u}_j$, often the approximate midpoint
Compute a point-valued approximate solution

$$\hat{u}_j(t) := \tilde{u}_j + \tilde{u}'_j(t - t_j) + \dots + \frac{\tilde{u}_j^{(p-1)}}{(p-1)!}(t - t_j)^{p-1}$$

Derivatives are computed by automatic differentiation

For Lorenz system,

Approximate Solution :=

$$\left(\begin{array}{l} 10 + 850t^2 - \frac{39050}{9}t^3 + \frac{956075}{54}t^4 - \frac{17867645}{162}t^5 \\ 10 + \frac{220}{3}t + \frac{6770}{9}t^2 + \frac{53390}{81}t^3 + \frac{15404365}{486}t^4 \\ - \frac{409237189}{1458}t^5 \\ 10 + 170t - \frac{1355}{3}t^2 + \frac{74065}{27}t^3 - \frac{12131195}{324}t^4 \\ - \frac{54827083}{972}t^5 \end{array} \right)$$

ODE: Step II.2. Enclosure \mathbf{z}_{j+1} of local error at t_{j+1}

True solution to $u' = f(t, u)$, $u(t_j) = \tilde{u}_j$ has local truncation error

$$\frac{1}{p!}u^{(p)}(\tau)(t_{j+1} - t_j)^p \subseteq \frac{1}{p!}\mathbf{u}^{(p)}(T_j)h^p$$

AWA computes \mathbf{z}_{j+1} as an enclosure of $\mathbf{u}^{(p)}(T_j)h^p/p!$ by evaluating the recurrence relation from the ODE in interval arithmetic

For Lorenz system,

$$[z_{j+1}] := \left(\begin{array}{l} [-0.21679335263710^{-8}, 0.22370921935310^{-8}] \\ [-0.99561703885010^{-8}, 0.12391430211310^{-7}] \\ [-0.11830168911410^{-7}, 0.10123356032310^{-7}] \end{array} \right)$$

ODE: Step II.3. Enclosure for transformation matrix \mathbf{A}_j

AWA views the true solution

$$u_{j+1} = u_{j+1}(z_0, z_1, \dots, z_{j+1})$$

as a function depending on all of the previously committed local errors

z_0, z_1, \dots, z_{j+1} are the exact (but unknown) local errors committed at each step

Expand u_{j+1} by the Mean Value Theorem:

$$u_{j+1} = \hat{u}_j(t_{j+1}) + \sum_{k=0}^{j+1} \frac{\partial u_{j+1}}{\partial z_k}(\hat{z}) \cdot (z_k - s_k),$$

where $\hat{u}_j(t_{j+1}) = u_{j+1}(s_0, s_1, \dots, s_{j+1})$

We want $\tilde{u}_{j+1} := \text{midpoint}([u_{j+1}])$, which we compute later

Hence, $\tilde{u}_{j+1} = \hat{u}_j(t_{j+1}) + s_{j+1}$, and $s_{j+1} := \tilde{u}_{j+1} - \hat{u}_j(t_{j+1})$

Partial derivatives are computed recursively. Let

$$\begin{aligned} A_{j+1,k} &:= \frac{\partial u_{j+1}}{\partial z_k}, \text{ for } k \leq j+1 \\ A_{j+1,j+1} &:= I \\ A_j &:= \frac{\partial u_{j+1}}{\partial z_j} \\ A_{j+1,k} &:= \frac{\partial u_{j+1}}{\partial z_k} = \frac{\partial u_{j+1}}{\partial u_j} \cdot \frac{\partial u_j}{\partial z_k} = A_j \cdot A_{j,j} = A_j A_{j-1} \cdots A_j \end{aligned}$$

ODE: Step II.3. Enclosure for transformation matrix \mathbf{A}_j

Then we have

$$\begin{aligned} u_{j+1} &= \hat{u}_j(t_{j+1}) + \sum_{k=0}^{j+1} \frac{\partial u_{j+1}}{\partial z_k}(\hat{z}) \cdot (z_k - s_k) \\ &= \hat{u}_j(t_{j+1}) + \sum_{k=0}^j A_{j+1,k}(z_k - s_k) + z_{j+1} \\ &= \hat{u}_j(t_{j+1}) + A_j A_{j-1} \cdots A_0(z_0 - s_0) \\ &\quad + A_j A_{j-1} \cdots A_1(z_1 - s_1) \\ &\quad \vdots \\ &\quad + A_j(z_j - s_j) \\ &\quad + I(z_{j+1} - s_{j+1}) \end{aligned}$$

Exact, but unknown values. To compute, apply enclosures
Let $y(t) \in R^{n \times n}$ be the solution to the variational equation

$$y' = \frac{\partial f}{\partial u} \cdot y, \quad y(t_j) = I$$

Partial derivative $\frac{\partial f}{\partial u}$ is evaluated along the true solution of $u' = f(t, u)$. Then

$$A_j = y(t_j) + y'(t_j)h + \dots + \frac{y^{(p-1)}(t_j)}{(p-1)!}h^{p-1},$$

a truncated Taylor series with no remainder

ODE: Step II.3. Enclosure for transformation matrix A_j

For an enclosure of A_j , solve the $n + n \times n$ -dimensional ODE system

$$\begin{aligned} u' &= f(t, u), \quad u(t_j) = \mathbf{u}_j \\ y' &= \frac{\partial f}{\partial u}(t, u) \cdot y, \quad y(t_j) = I \end{aligned}$$

Solve one $n + n \times n$ -dimensional ODE system, or
Solve one n -dimensional followed by n n -dimensional copies
of the same system with different initial conditions
For Lorenz system,

$$\begin{aligned} x'(t) &= 10y(t) - 10x(t) \\ y'(t) &= 28x(t) - y(t) - x(t)z(t) \\ z'(t) &= x(t)y(t) - \frac{8}{3}z(t) \\ Y1'(t) &= 10Y2(t) - 10Y1(t) \\ Y2'(t) &= 28Y1(t) - Y1(t)z(t) - Y2(t) - x(t)Y3(t) \\ Y3'(t) &= y(t)Y1(t) + x(t)Y2(t) - \frac{8}{3}Y3(t) \end{aligned}$$

ODE: Step II.3. Enclosure for transformation matrix A_j

For the first set of initial conditions, the solution is

$$\begin{aligned} x(t) &= 10 + 850t^2 - \frac{39050}{9}t^3 + \frac{956075}{54}t^4 \\ &\quad - \frac{17867645}{162}t^5 + O(t^6) \\ y(t) &= 10 + 170t - \frac{1355}{3}t^2 + \frac{74065}{27}t^3 - \frac{12131195}{324}t^4 \\ &\quad - \frac{54827083}{972}t^5 + O(t^6) \\ z(t) &= 10 + \frac{220}{3}t + \frac{6770}{9}t^2 + \frac{53390}{81}t^3 + \frac{15404365}{486}t^4 \\ &\quad - \frac{409237189}{1458}t^5 + O(t^6) \end{aligned}$$

$$Y1(t) = 1 - 10t + 140t^2 - \frac{9770}{9}t^3 + \frac{217205}{54}t^4$$

$$- \frac{3107903}{162}t^5 + O(t^6)$$

$$Y2(t) = 18t - \frac{557}{3}t^2 + \frac{14131}{27}t^3 - \frac{1804673}{324}t^4$$

$$- \frac{128952941}{4860}t^5 + O(t^6)$$

$$Y3(t) = 10t + \frac{335}{3}t^2 - \frac{26155}{27}t^3 + \frac{3509015}{324}t^4$$

$$- \frac{115264679}{972}t^5 + O(t^6)$$

Compute A by evaluating at $t = t_{j+1}$:

$$A_0 = \begin{pmatrix} 0.975858193 & 0.0246611249 & -0.000309004582 \\ 0.0438475408 & 0.997742160 & -0.0248920560 \\ 0.0256831921 & 0.0252097623 & 0.993041869 \end{pmatrix}$$

AWA computes an interval enclosure of A , but we will settle for Maple's result.

ODE: Step II.4. Prepare for next solution step

$$\begin{aligned} u(t) \in & \left(\begin{array}{l} \text{point-valued truncated Taylor} \\ \text{series for approximate solution} \end{array} \right) \\ & + \left(\begin{array}{l} \text{old} \\ \text{error} \end{array} \right) + \left(\begin{array}{l} \text{enclosure of local} \\ \text{truncation error} \end{array} \right) \end{aligned}$$

At $t_{j+1} = 0.0025$ we assemble the solution enclosure from

$$s_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$z_0 = \begin{pmatrix} [0, 0] \\ [0, 0] \\ [0, 0] \end{pmatrix}$$

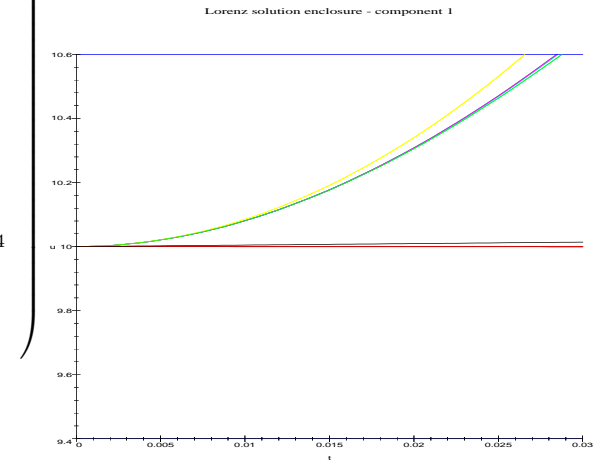
$$s_{j+1} := \tilde{u}_{j+1} - \hat{u}_j(t_{j+1}) = \begin{pmatrix} 0.34610^{-10} \\ 0.1217610^{-8} \\ -0.853410^{-9} \end{pmatrix}$$

ApproximateSolution =

$$\begin{pmatrix} x(t) = 10 + 850t^2 - \frac{39050}{9}t^3 + \frac{956075}{54}t^4 \\ -\frac{17867645}{162}t^5 + O(t^6) \\ y(t) = 10 + 170t - \frac{1355}{3}t^2 + \frac{74065}{27}t^3 - \frac{12131195}{324}t^4 \\ -\frac{54827083}{972}t^5 + O(t^6) \\ z(t) = 10 + \frac{220}{3}t + \frac{6770}{9}t^2 + \frac{53390}{81}t^3 + \frac{15404365}{486}t^4 \\ -\frac{409237189}{1458}t^5 + O(t^6) \\ \hat{u}_j(t_{j+1}) = \begin{pmatrix} 10.0052453856954 \\ 10.1880462319424 \\ 10.4222184769381 \end{pmatrix} \end{pmatrix}$$

We are now ready to go on to the next integration time step.

ODE: Fig A. Enclosure of Lorenz component 1

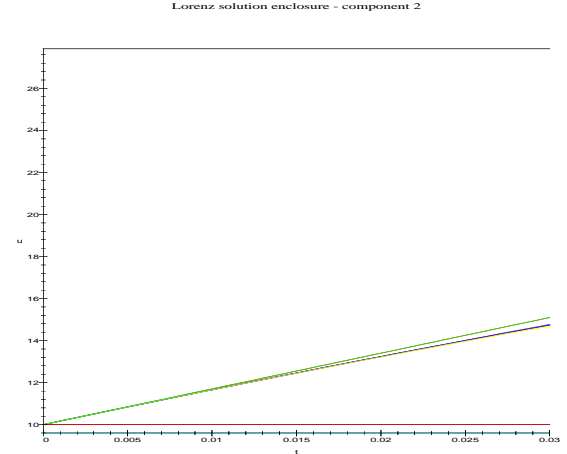


$$[z_{j+1}] = \begin{pmatrix} [-0.21679335263710^{-8}, 0.22370921935310^{-8}] \\ [-0.99561703885010^{-8}, 0.12391430211310^{-7}] \\ [-0.11830168911410^{-7}, 0.10123356032310^{-7}] \end{pmatrix}$$

ODE: Fig B. Enclosure of Lorenz component 2

$$A_0 = \begin{pmatrix} 0.975858193 & 0.0246611249 & -0.000309004582 \\ 0.0438475408 & 0.997742160 & -0.0248920560 \\ 0.0256831921 & 0.0252097623 & 0.993041869 \end{pmatrix}$$

$$\begin{aligned} [u_{j+1}] &= \hat{u}_j(t_{j+1}) + A_j A_{j-1} \cdots A_0 (z_0 - s_0) \\ &\quad + A_j A_{j-1} \cdots A_1 (z_1 - s_1) \\ &\quad \vdots \\ &\quad + A_j (z_j - s_j) \\ &\quad + I(z_{j+1} - s_{j+1}) \\ &= \begin{pmatrix} [10.0052453835274, 10.0052453879326] \\ [10.1880462219861, 10.1880462443339] \\ [10.4222184651078, 10.4222184870616] \end{pmatrix} \end{aligned}$$



ODE: Fig C. Enclosure of Lorenz component 3

How accurate (tight) are the enclosures?

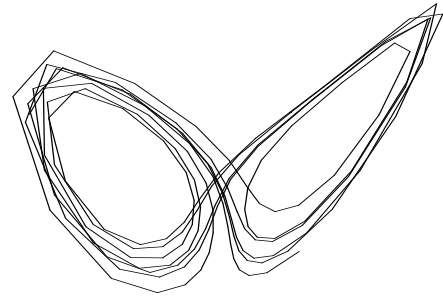
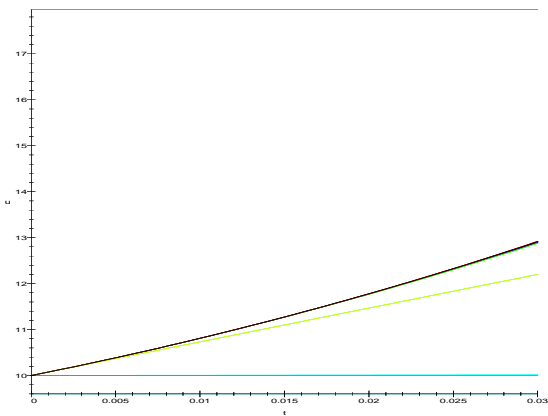
$$\text{width}([u_{j+1}]) = (0.4405210^{-8}, 0.22347810^{-7}, 0.21953810^{-7})$$

Enclosures DO remain within the *a priori* bounds

Base for the approximate solution on the next time step:

$$\tilde{u}_{j+1} := \text{midpoint}([u_{j+1}]) = \begin{pmatrix} 10.0052453857300 \\ 10.1880462331600 \\ 10.4222184760847 \end{pmatrix}$$

Lorenz solution enclosure - component 3

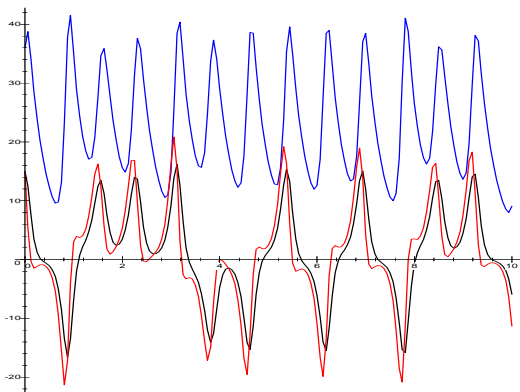


ODE: AWA solution of Lorenz system

Call AWA for $t \in [0, 10]$, $h = 0.0156$
 Largest width of solution enclosure $\approx 10^{-7}$
 Plot midpoints of enclosing intervals

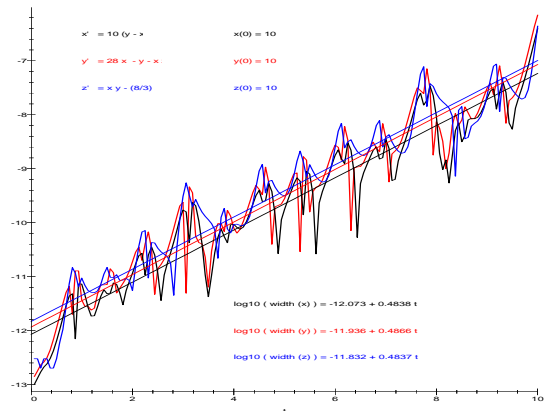
ODE: Fig H. Individual components of Lorenz solutions

Individual components of Lorenz solutions



ODE: Fig I. Widths of enclosures of Lorenz solutions

Widths of enclosures of Lorenz solutions



ODE: Fig J. Lorenz attractor computed to $t = 10$ by AWA